University of Twente
Enschede - The Netherlands

A stable matching based adaptive subcarrier assignment method for multimodal fibre access networks

Master's thesis

Author:
Bart Sikkes

Supervising committee:
Prof.ir. A.C. van Bochove
Dr.ir. S.M. Heemstra de Groot
R.O. Taniman M.Sc.

Design and analysis of communication systems
Faculty of electrical engineering, mathematics and computer science
University of Twente
Augustus 1, 2006

Abstract

We live in a time when the internet becomes more and more present in our lives and other services start using the same infrastructure. This causes an ever increasing need for bandwidth, both in core networks and access networks. When focussing on the latter we see a shift from using copper to using glass. These networks bring new challenges to overcome. For example the subcarrier assignment problem when using the higher order lobes of a multimodal fibre.

Developing an adaptive subcarrier assignment method to work in such a network is the goal of this master assignment. To do this, first similar problems and algorithms are explored. During that exploration we came across the stable matching algorithm. To determine how well this algorithm works we need to test it. But as the real system isn't available yet, a simulation has to be used.

Once the simulation is implemented the functionality of the algorithm will be tested and we will determine how well it performs compared to other subcarrier assignment methods. For this three other methods are introduced: contiguous, interleaved and Hungarian algorithm based.

The four methods are then compared on two things: on the amount of bits the ONUs are able to load per symbol and on the mean sojourn times the ONUs experience. Several cases of simulation inputs are defined to explore the different aspects of the stable matching method.

These simulations show that indeed a working adaptive subcarrier assignment method based on the stable matching algorithm has been developed. This resulting method has some useful properties which are discussed in this thesis.

Preface

This thesis is the result of my master's assignment done at the Design and Analysis of Communication Systems (DACS) chair at the University of Twente. Most of the work was performed at the DACS laboratory.

I'd like to thank my supervisors R.O. Taniman M.Sc., Prof.ir. A.C. van Bochove and Dr.ir. S.M. Heemstra de Groot. Their support and guidance throughout this master's assignment has helped a lot.

Finally I'd like to thank my fellow students at DACS, flatmates, friends and family for their support and accepting the occasional verbal explosion.

Table of contents

1 Introduction

1.1 Motivation

Because of continuing growth of interest in services like internet, digital television and triple play there is a constant need for more bandwidth. While technologies like ADSL 2+ and VDSL try to squeeze the maximum amount of bits from a copper wire, fibre is the most likely way to reach real high bandwidths. Currently most parts of the access networks are already using fibre, but the so called “first mile” (previously known as “last mile”) is often still copper. This is slowly changing and therefore more research is being done in the area of fibre in the first mile.

Recent research (see for example [RaWh99] and [KoBo03]) has proven that not only the baseband of the multi-mode fibre can be used for data transmission, but also the higher order lobes of the spectrum. The problem with using this part of the fibre spectrum is that it is expected to be time-variant and user specific. The part of the spectrum containing these higher order lobes is divided in a number of smaller sections containing modulated subcarriers. To be able to effectively use this part of the spectrum a so-called adaptive subcarrier assignment method based on channel quality is needed.

Such a method will assign available subcarriers to users based on the quality of a subcarrier for a user. Next to that it would be nice if the algorithm takes the traffic characteristics of the users into account while running. And when the channel quality or traffic characteristics change, the method should adapt to these changes and reassign subcarriers if required. The development and simulation of such a method is what is described in this thesis.

This master assignment is part of a larger project from the IOP GenCom program called Full-service Access Network using Multimode Fibre [IOP-GenCom]. On the project the DACS group at University of Twente cooperates with the ECO group at the University of Eindhoven. The goal of the project is to define, design and implement a multimode fibre access network for FTTH.

1.2 Problem definition

The design, simulation and analysis of an adaptive subcarrier assignment method is what this assignment is about. To keep the assignment manageable some limitations have been introduced.

The subcarriers are assigned to users based on the quality of a subcarrier for a user and the traffic characteristics for that user. At first it was envisioned that research would be started in the area of channel quality via Signal to Noise Ratio (SNR) estimation. After

some initial research this appeared to be out of scope for the assignment. It is now assumed that full channel information is known at the central office and at the users.

As mentioned, this assignment is a subpart of the project in which the full service access network is designed and implemented. Currently that project is in its definition and early design stage, so a complete system isn't available yet. To evaluate expected results, a simulator will be used. In this simulation the behaviour of a fibre will be modeled to be used as the input for the model.

1.3 Research goal

Based on the above problem definition we define the following main goal of the assignment: develop and simulate an adaptive subcarrier assignment method to be used in a full service access network using multimode fibre.

To reach this goal several subgoals have been determined:

- Research the current subcarrier assignment methods in similar systems.
- Define subcarrier assignment method.
- Implement subcarrier assignment method in a simulation.
- Evaluate subcarrier assignment method.

1.4 Approach

The assignment is roughly executed as follows.

Research
The research starts with some initial papers and information received from my supervisors. Both the content and references from the papers contain information to work with. Next to that, different internet search engines have been used to gather information: specialistic search engines like Springer online [Springer] and general ones like Google [Google]. The first subgoal is to find general information on subcarrier assignment. After that, the next subgoal is to create an overview of the currently used subcarrier assignment algorithms and what techniques are used.

Defining
Here the subcarrier assignment method will be defined by first determining which technique is most suitable and then actually designing the method.

Implementing
Once the method has been defined it will have to be implemented in a simulation so it can be executed and then evaluated.

Evaluating
When the method has been implemented it has to be evaluated to determine if it works like designed and to determine how well it function compared to other methods.

1.5 Arrangement of the next chapters

In chapter 2 the full-service access network in which the subcarrier assignment algorithm will be used is described. The technologies used in the network and their effect on this assignment are also discussed.

Chapter 3 gives the basic principles of subcarrier assignment. Further different types of subcarrier assignment methods are described. Following in chapter 4, the algorithms used in the chosen subcarrier assignment methods, stable matching and the Hungarian algorithm, are described.

In chapter 5 the implementation of the simulation and assignment methods is presented and the inputs and outputs from the simulations are described. Also the results which have been obtained by executing the simulator are given and discussed.

In the final chapter 6 the conclusion is presented and possible future work is discussed.

2 Related systems and technologies

As mentioned in the introduction this master assignment is a sub goal of a larger project that has as goal to create a fibre-to-the-home (FTTH) access network based on multimodal fibre. In this chapter we will discuss some of the technologies and systems used in that project to place this master assignment in its proper context.

2.1 FTTH

Fibre-to-the-home is the term for optical networks used in the so called “first mile”. That is the area between neighbourhood access points and the actual homes of the customers. Usually fibre is already used up to this first mile, and from there copper is used. As systems like ADSL appear to be unable to achieve the really high bandwidth that is expected to be required in the future, FTTH is gaining popularity.

Two approaches are used on deciding what type of components to use in the network; active and passive. In the case of active there will be electronic equipment installed in the neighbourhood that does switching and routing. With the passive variant, which is commonly called PON, this is not the case. Because this passive approach is used in the larger project it will be discussed more in depth now.

2.2 PON

PON stands for passive optical network. As the name says all components in the optical distribution network are passive. This in opposite to active components. It means that the components don’t require any power and don’t process the information on the network. The main advantage of such a network is that the components are simpler and are expected to last longer than active components; which means less servicing and repairing. Another advantage is that the network can just be put in the ground without needing external power.

A PON, from which an example can be seen in Figure 2.2, consists of a central controlling component at the Central Office (CO) called Optical Line Terminator (OLT), this is the start point of the PON. From this OLT there

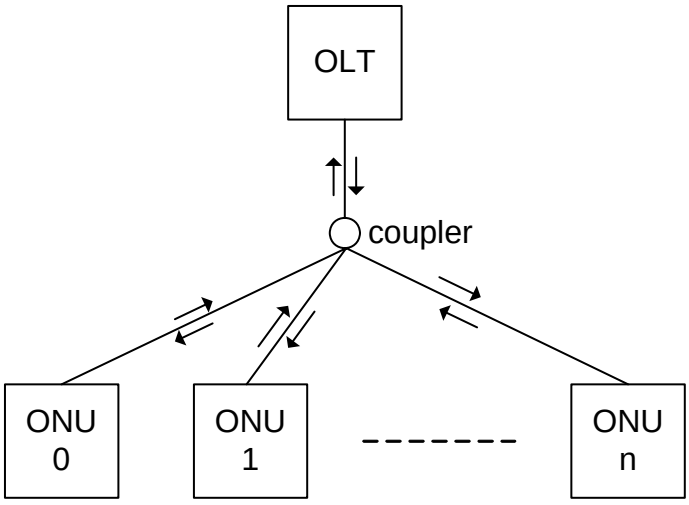

Figure 2.1: A PON with n ONUs

will be fibre to all the end nodes of the PON called Optical Network Units (ONU), these are also called Optical Network Terminals (ONT). To have the same signal received by all ONUs a coupler or splitter will be used between the initial signal from the OLT and all the ONUs.

2.3 The fibre

The type of fibre being used in the PON will be a so called multimode fibre. This is a type op fibre with a large core diameter then the more often used single mode fibre. Because of this larger core diameter it is possible to have more modes propagating through it, therefore the name multimode fibre. This larger core has certain advantages, the main one being that multimode fibres are easier to handle. The main problem is that their bandwidth * fibre length ratio is less then with single mode fibre.

Research has been done to see if there are any ways to overcome this problem. One of the things that was discovered is that a higher frequency area of the fibre spectrum appears to be useable for transmitting signals. In the larger project it will be attempted to make use these higher order lobes to make multimode fibres a viable transmission medium for the access network. Unfortunately these higher order lobes have a problem also. The power spectrum is time variant and different for each user of the system. Therefore a subcarrier assignment algorithm has to be created that takes into account these issues, and that is what this thesis is about.

2.4 Communication in a PON

There are two directions in which traffic can flow in a PON: the downstream direction, in which the traffic goes from the OLT to the ONUs and the upstream direction, in which the traffic goes from the ONUs to the OLT.

The handling of downstream traffic is usually rather simple to implement. When looking at the most common scheme used in PONs, single channel with TDMA (Time Division Multiple Access), in the downlink direction the data for all ONUs is send from the OLT towards the ONUs. Then at the passive splitter the data is “replicated” and send to all the ONUs. At the ONU the data meant for that ONU is accepted and the rest is discarded.

In such a PON in the upstream direction things become more complicated. There has to be some scheme to prevent collisions when the data from the different ONUs arrive at the OLT receiver at the same time. For this, a time-division multiplexing approach with synchronized timeslots can be used. Access to the uplink channel is then divided in timeslots. Every ONU has it own timeslot in which it can send data and it doesn’t send during the timeslots of the other ONUs.

As mentioned before in the system that will be considered in this thesis, subcarrier multiplexing is used. In the downstream direction this means that each ONU will only

accept the data from the set of subcarriers it is assigned. For the handling of upstream traffic it is extra useful, because a non overlapping subset of subcarriers is used by just one ONU in a time epoch thus preventing the necessity of time division-multiplexing..

As transmissions between the OLT and ONUs should be in both directions (duplex), a certain scheme must be used for that purpose. This is done by separating the traffic in the upstream and downstream direction in some way. One possibility is by using separate fibres for the upstream and downstream traffic. Another option is using different wavelengths for upstream and downstream traffic. In the system considered in this thesis, we assume some duplexing scheme is in place without specifying how that is done exactly. A relevant reference can be given for this matter which is [TaBo05].

3 The subcarrier assignment problem

In this chapter we will present the subcarrier assignment problem. And discuss solutions that have been developed for it.

In a subcarrier multiplexing scheme, especially the multicarrier variant one, there is a number of subcarriers available and a number of ONUs that want to use those subcarrier to transmit data. To make this possible the subcarriers will have to be assigned to the ONUs for a certain amount of time. That amount of time is called the time epoch, it can vary from a large number of short time periods in a quickly changing environment or one long time period. Theoretically these assignments are possible in every combination between ONUs and subcarriers. In practice it will depend on which subcarrier is best suited for an ONU, or which set of subcarriers is best suited for an ONU.

During subcarrier assignment the subcarriers are assigned to achieve some goal. Common goals are: maximize the overall throughput or minimize the overall power consumption. This can be combined with per ONU requirements, for example every ONU has at least a certain minimum throughput available.

In wireless systems it is usually attempted to minimize the power, in the system under study here it is attempted to maximize the throughput. So we attempt to assign a number of subcarriers to a number of ONUs within a certain total power limit, to prevent clipping, trying to maximize the total throughput while trying to stay below a certain BER.

Next to subcarrier the term subchannel is used in the chapter. A subchannel is a further undefined entity to transport information in an environment with multiple channels. One way to do that is with subcarriers.

3.1 Subcarrier assignment method

Initially we will look at one ONU and several available subcarriers, later on we will discuss the effect of multiple ONUs.

3.1.1 Single ONU

A subcarrier assignment method calculates an optimal subcarrier assignment based on certain inputs and the goals it is designed to achieve. The inputs can for example be the subchannel characteristics and the traffic characteristics. The first describes the quality of the subchannel, e.g. in form of the subchannel normalized SNR and the second describes the how much data an ONU wants to send (expressed in its queue size). The output is of course the subcarrier assignment.

Once the subcarrier assignment is done another step has to be taken before data can actually be transmitted. For each subcarrier it has to be determined how many bits are available per symbol, this is called the bit loading. During bit loading the amount of bits per modulation symbol is determined based on, for example, channel conditions (the normalized SNR), available power and required BER. Before bit loading can be performed a subcarrier assignment should be available.

3.1.2 Multiple ONUs

With a single ONU and several subcarriers to choose from, the subcarrier assignment isn't that hard to get. The subcarrier assignment method picks the best subcarriers while satisfying its constraint and is done. When multiple ONUs are considered things become more complicated. Now when one subcarrier is assigned to a certain ONU it can't be assigned anymore to one of the other ONUs. A simple approach would be for the subcarrier assignment method to determine for every subcarrier what the best ONU is and make those assignments. But this could cause an assignment where some ONUs don't get any subcarriers.

This isn't a problem when the system doesn't care that some of its ONUs might never be able to send any data. But it is reasonable to assume that in every system with a subcarrier assignment method one wants to make sure every ONU can send data. So a method will always have to make sure every ONU gets some of the available subcarriers and is able to load some bits on those.

Making sure every ONU gets some throughput is still relatively easy. Things become more complicated when it is attempted to achieve, for example, a maximum overall throughput. Now it isn't possible anymore to just assign a certain subcarrier to a certain ONU without looking at the other ONUs, as that might cause a non optimal assignment.

The most straight forward way to deal with this is to generate every possible assignment and then check which assignment offers the best overall throughput. The problem with that is that it will take many calculations to determine all possible assignment. Moreover, the algorithm doesn't just have to combine all subcarriers with all ONUs, but also do the bit loading. Next to that it might be required to do the calculation again if the related circumstances change, for example, the subchannel quality, causing even more work. So in general this calculation takes too much time, generally with exponential complexity, and another solution must be found.

As achieving a maximum (per ONU or just overall) is something that would be very useful there have been many attempts to find a way to do it. As these attempts are closely related to a certain general approach of subcarrier assignment they will be discussed in the next section about different approaches to subcarrier assignment.

3.2 The methods

How the subcarrier assignment method will achieve its goals depends on the type of method. In this section some approaches to subcarrier assignment methods will be discussed.

3.2.1 Heuristic approach

In a heuristic approach, it's common to make use of known optimal algorithms for similar types of problems. These can then be combined and often small additions / modifications are made to create a new approach. An example of its use is when one divides a problem into parts and then uses a known optimal algorithm on one or more of those parts.

One possibility is to use recent ideas from the research on multiuser orthogonal frequency-division multiplexing (OFDM), also known as orthogonal frequency division multiple access (OFDMA). OFDM is a modulation system in which the whole frequency band is divided into subchannels that are orthogonal to each other and different modulation level (of QAM) can be used for different subchannels. The OFDMA version adds the support for multiple users.

It's a typical fact that the research in the multiuser area is often a continuation of research done on single user OFDM. A regularly used subcarrier assignment approach in single user OFDM is the water filling algorithm. This algorithm has been extended to the case of multiuser OFDM.

3.2.2 Water filling

The single user water filling, also known as water pouring algorithm was designed to do power allocation. It is assumed that one user wants to transmit data and has a number of subchannels available. It is also assumed that the subchannel characteristics are known. The question is how to optimally spend the power the user has available.

According to the water filling theorem the optimal way is to use the subchannels with the

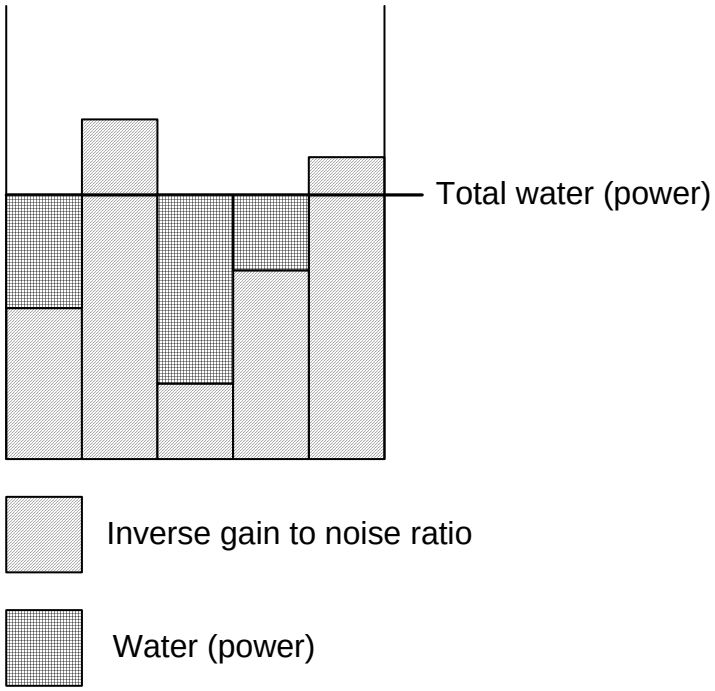

Figure 3.1: Water filling

lowest inverse gain to noise ratio. The power will then be allocated as if it is water. For this one can picture the gain to noise ratio as a bowl which is being filled with water (power). The higher sides of the bowl won't be filled as there isn't enough water (power) for that. A picture of the result of the water filling algorithm can be seen in Figure 3.2. It consists of five subchannels with the associated inverse gain to noise ratios and shows that three subchannels will be assigned a certain amount of power and two have a too bad ratio.

There are several variations on the original water filling algorithm, which sometimes is called the true or classical water filling algorithm. These variations are often relaxations on some of the requirements in the original algorithm. That way it is possible to design simpler and therefore less demanding algorithms. This is something that can be very useful when the algorithm needs to run on a limited processor power system.

One of these variations, called constant power water filling [YuCi01], focuses on the power allocation. In the classic water filling algorithm this is done per subchannel based on the Shannon's Gaussian capacity formula. It has been shown that this exact power allocation isn't very important. An almost identical result can be gotten when constant power allocation is used. First is determined which of the subchannels are allocated power. That is done based on the gain levels of the subchannels. Then the chosen ones all get the same amount of power.

3.2.3 Multiuser water filling

In a multiuser environment there is more then one user that wants to transmit over a number of subchannels and there is an amount of power available per user. Now it isn't possible anymore to just let every user do water filling for itself. Because that might cause problems for other users, they could end up with subchannels they can't transmit any data on.

A possible approach is discussed in a paper by G. Münz, et al. [MuPf02]. Here the water filling theorem is used to do the subcarrier assignment for multiple users. The water filling graphs for all users are combined and best ones are chosen for each user. When the subcarrier assignment is finished then, on a single user basis, the power allocation and bit loading is done by already established algorithms. This can be done, for example, with the single user water filling approach.

3.2.4 Linear programming approach

Another mathematical field that can offer assistance with subcarrier assignment is the operations research, shortened to OR. It was initially started to help the British armed forces during World War II with certain logistics and training problems. After the war the use spread to other areas, most often in industrial applications. Currently OR is seen as an scientific approach to decision making, which seeks to determine how best to design and

operate a system, usually under conditions requiring the allocation of scarce resources [Wins04].

Linear programming is one of the systems used to find optimal solutions to OR problems. This requires the problem to be written in the linear programming model form. Such a form always consists of the same structure and makes it possible to more easily determine the optimal solution. The linear part of the name relates to the used functions in the model, they have to be linear.

First the required decision variables need to be determined. These describe the decisions to be made by the model. The object function consists of the decision variables with their coefficients. This function will be maximized or minimized according to what the system is trying to achieve. For example, if the object function describes costs, it will be minimized. But if it's about profit then it will of course be maximized.

If there would be just an object function then optimizing it would go on till infinity. This is never the case as there are always some restrictions; these are called constraints in a linear programming problem. A constraint limits the value of a decision variable. Finally there are sign restrictions that are used to express if a decision variable can assume negative and positive values or just positive ones.

A typical linear programming problem looks like this:

```
Max z =   c1x1 +    c2x2           (object function)
          x1   +    x2   < 10      (constraint 1)
                    x2   < 5       (constraint 2)
          x1             > 0       (sign restriction 1)
          x2             > 0       (sign restriction 2)
```

There are some variations of the basic linear programming problem. One of the most common ones is the integer only version, where the solution is only allowed to consist of integer values.

3.2.4.1 Applying to subcarrier assignment

So can OR / linear programming be applied to subcarrier assignment? We certainly want to optimize something, for example the throughput or the power consumption. Examples of the use of linear programming in subcarrier assignment can be found in the literature [KiLe01], [RhCi01]. In several papers the linear programming form is used, but then just to express the problem. Other approaches are then used to design a subcarrier assignment algorithm. In other cases it is attempted to actually solve the linear programming problem via OR techniques.

3.2.5 Stable matching

Another approach comes from the stable matching problem, which on first glance doesn't appear to be an obvious choice. But with some extra code around it can be used very well for a subcarrier assignment method. It is able to take two aspects into account during the execution and implementing it into computer code seems straight forward. For more information on this method we point to the first part of chapter 4.

3.2.6 Comparing the methods

In the previous sections we have seen a number of possible solutions to the subcarrier assignment problem. Now we have to determine which one(s) can be useful for us in the previously described network.

Single user water filling is certainly not an option, as we have a multi user environment, though the multiuser variant might be useful. It results in an overall optimal subcarrier assignment on which then power allocation and bit loading has to be performed. The method only uses the gain as input, no other aspects are used. When a certain user has a bad gain compared to the others it can end up without any subcarriers.

As mentioned before in the section on linear programming a mapping to an assignment problem is possible. By using existing solving techniques for linear programming it should then be possible to determine and optimal solution. With adding enough constraints we should be able to form the solution we want. The problem is that we would want to use integer linear programming (i.e. we don't want to assign half subcarriers) and solving such problems is NP hard.

The stable matching approach is able to come with a solution that looks at the interests of the individual entities and won't result in a user getting no subcarriers at all. Further it is able to take two aspects into account, something which was mentioned earlier as an interesting extra feature. There is a problem though; its basic version doesn't allow multiple entities to be matched with one. As we will have more subcarriers than users and want all of them assigned, this certainly is an issue.

After looking at these different aspects of these methods we feel that the stable matching approach is a promising way to go for the subcarrier assignment method we want to develop. This because of the inherent fairness and that it can take two aspects into account. Of course the issue of not being able to match multiple entities to one has to be taken care of.

4 The used algorithms

In this chapter we will present the algorithms that are used in the subcarrier assignment methods which behaviour is simulated in chapter 5. We start with the stable matching algorithm and then present the Hungarian algorithm.

4.1 Stable matching

The stable matching algorithm is the basis of our main assignment method. In this section we start by giving some brief history on the algorithm, then present the basic form and continue with some extensions on it. We conclude this section with describing what we did with the stable matching algorithm to make it suitable for use in a subcarrier assignment method.

4.1.1 Introduction to stable matching

Formally the stable matching problems where introduced by Gale and Shapley in their paper [GaSh62] on stable marriage problems. We can view stable matching problems as a group of problems which contains different types of stable matching problems, for example the stable marriage problem. These problems differ in some aspects that we will discuss later on, but they all want to result in a stable matching.

A basic matching is the situation when two sets containing certain entities have been connected to each other. An example can be seen in Figure 4.2.

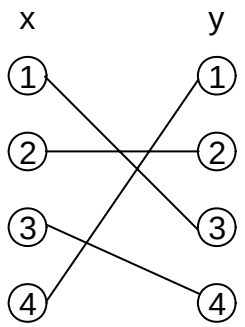

Figure 4.1: A basic matching

Before we can continue with discussing stable matching we have to introduce the preference lists. Every entity (so all from both sets) puts all the entities in the other set in a certain order, this called the preference list. Now we can call a matching stable when none of the entities rather want to be connected with another entity then their currently matched entity.

In the stable marriage problem the two sets are equal sized and consist of men and women. The result should be a one-on-one mapping (marriages) in which all men and women are married and satisfy the previously mentioned stability criterion.

In the hospitals/residents (also called college admissions problem) problem there are two sets, one contains the residents and the other contains the hospitals. Here each hospital has a certain maximum number of places available for residents. This means that each

hospital can be matched with more then one resident, up to the maximum number of places. Each resident is just matched with one hospital

In their paper Gale and Shapley presented an algorithm to solve the stable marriage problem. Using the algorithm, it can be showed that for each stable marriage problem, so for every combination of preference lists, there is at least one stable result. The result has a special property; it gives the men the best possible partner within any stable matching. In the same paper this algorithm was generalized to handle the hospitals/residents problem also with the same property.

4.1.2 Basic stable matching algorithm

In the form we describe here we call the stable matching algorithm the stable marriage algorithm. There are two disjoint sets of men and women of an equal size N. Every man has a preference list containing all women in the order he likes them best. Every woman has a similar list but then with all the men. When a man and woman are matched we call them partners. If a certain pair of a man and a woman who are in the current matching not partners but favour each other instead of the current partners exists in the matching we call that a blocking pair. The existence of such a blocking pair means the current matching is unstable.

We now present the basic Gale Shapely algorithm as noted in the book The Stable Marriage Problem, Structures and Algorithms by D. Gusfield and R.W. Irving [GuIr89].

```
assign each person to be free;
while some man m is free do
begin
      w:= first woman on m's list to whom m has not yet proposed;
      if w is free then
            assign m and w to be engaged
      else
            if w prefers m to her fiancé m' then
                  assign m and w to be engaged and m' to be free
            else
                  w rejects m
end;
output the stable matching consisting of the n engaged pairs
```

The principle is simple, a free man goes through his preference list proposing to the first woman he hasn't proposed to yet. If the woman being proposed is free they become engaged. If the woman is already engaged but favours him over her current partner, they become engaged and the previous partner is free again. If the woman is engaged and doesn't like the proposing man better then her current partner, she will reject the proposing man. This process continues until all men have been engaged, then the engagements can be turned into marriages.

Let's look at a very simple example with three man and woman. First we show the preference lists for the man and woman in tables below. So man 1 likes woman 2 the best, then woman 1 and woman 3 the least.

Man	1st	2nd	3rd
1	2	1	3
2	3	2	1
3	2	3	1

Woman	1st	2nd	3rd
1	1	2	3
2	3	1	2
3	1	3	2

When the algorithm is executed the following steps are made:

- man 1 proposes to the first woman on his list, woman 2, she accepts.
- man 2 proposes to the first woman on his list, woman 3, she accepts.
- man 3 proposes to the first woman on his list, woman 2, she favours him over man 1 so she accepts the proposal from man 3 and breaks with man 1, he becomes free again.
- man 1 proposes to the second woman on his list, woman 1, she accepts.

This results in the following stable assignment.

Man	Woman
1	1
2	3
3	2

Two of important properties of the Gale-Shapely algorithm are that it terminates for every given preference lists and that it always produces a stable matching. We now show the proofs from the previously mentioned book that this is indeed the case.

First that it always terminates, for this we first show that no man can be rejected by all women. A man can only be rejected by an engaged woman and engaged women never become free again. So if a man would be rejected by the last woman on his list that means that all women are engaged. But as the number of man and woman is the same and no woman is allowed to be engaged with more then one this isn't possible. Further does every step in the algorithm relate to one proposal and no man proposes twice to the same woman. Therefore with N men and N women the number of steps can never be larger then N^2. Based on these two things we can be sure the algorithm terminates.

Secondly we show that the matching is always stable. If a man prefers a woman over his current partner then that woman must have rejected him during the execution of the algorithm. During that rejection the woman has been engaged to a man she prefers over the first man. Every other engagement she breaks to start a new one will only result in a better man for her. So this woman can't prefer the initial man over her current partner and they can't form a blocking pair. Using this reasoning on every man shows us that there isn't any block pair and that the matching is stable

As can be seen, this is the man-oriented version, every man will end up with the best possible woman for him within the stable matching. By letting the woman propose it is simple to turn it into a woman-oriented version.

The property that in the man-oriented version every man get's the best woman can be proved also, we do that now. This proof, again from [GuIr89] is a little more extensive then the previous one and it involves another property, that every execution of the Gale Shapely algorithm results in the same stable matching.

First we assume that *E* is an arbitrary execution of the algorithm and results in a stable matching *M*. Then we assume there is a different stable matching *M'*, this is in contradiction with the mentioned property. Now suppose there is a man *m* that prefers his partner woman *w'* from *M'* over his partner woman *w* from *M*. So during the execution *E*, *w'* must have rejected *m*. Assume that this was the first rejection of a stable partner during the execution and that it was caused by the engagement of *w'* with *m'*. This means that *m'* can't have a stable partner who he prefers over *w'*, as *w'* was the first woman to reject a stable partner. We now know that *m'* prefers *w'* over his current partner in *M'* and therefore they are the blocking pair in *M'*. From that we can conclude that every man *m* is matched with his favourite stable partner *w* in *M*. Also because *E* was an arbitrary execution of the algorithm this shows that all executions of the algorithm result to the same stable matching.

4.1.3 Extensions to the stable matching algorithm

Since its first appearance, there have appeared many variations from small to large to the original stable matching problem. Examples are: lying by entities, unacceptable partners and with three sets of entities. While these can be very useful and interesting in certain situations, we won't use them. We restrict this discussion of extensions to ones we can / will actually use.

4.1.3.1 Unequal set sizes

Previously we assumed that the sizes of the sets of man and woman are equal. But there are situations in which this isn't the case and we still want to have a stable matching. For a simple extension like this, there are several possibilities. In the previously mentioned book, they simply leave a part of the larger set unmatched. It is proved then that there is always at least one stable matching that contains all members of the smaller set. The larger set can be divided in two parts, one contains the members that are always matched, the other contains the ones that are never matched.

4.1.3.2 Hospital/Resident

The previous one is one possible solution to the unequal set sizes problem, but what if we want to match all members of the larger set? In the stable marriage version this means that we have to allow multiple women being partnered with one man. This problem is

very similar to the previously mentioned hospitals/residents problem. There, each hospital can accommodate one or more residents. The hospital-oriented algorithm from the previously mentioned book is shown below. We call an hospital undersubscribed when it has places available for residents. As with the stable matching version before the algorithm can be executed the hospital and residents make preference lists containing all the opposite entities.

```
assign each resident to be free
assign each hospital to be totally unsubscribed
while (some hospital h is undersubscribed) and (h's list contains a
      resident r not provisionally assigned to h) do
begin
      r := first such resident on h's list
      if r is already assigned, say to h', then
            break the provisional assignment of r to h'
      provisionally assign r to h
      for each successor h' of h on r's list do
            remove h' and r from each other's lists
end
```

This algorithm uses a different approach then the original Gale Shapely one. It removes entries from the preference lists when it is clear they won't be used later on. Once the algorithm finishes the undersubscribed hospitals have been assigned the residents on their reduced lists. The hospitals that are not undersubscribed have been assigned the first q residents on their reduced lists, where q is the number of available places.

4.1.3.3 Freeing entities

It turns out that a little different version of the Gale-Shapley algorithm we created can be used for this variant of the problem with a similar result as for the original stable marriage problem. Meaning there is always at least one stable matching. The algorithm is shown below.

```
assign each person to be free
while some man m is free do
begin
      w:= first woman on m's list to whom m has not yet proposed
      if w is free then
            assign m and w to be engaged
      else
            if w prefers m to her fiancé m' then
                  assign m and w to be engaged and m' to be free
            else
                  w rejects m
      if there are no free man and there are unassigned women then
            free the man
end
output the stable matching consisting of the n engaged pairs
```

The reason that every man only gets one woman in the stable marriage problem is that once a man and woman engage the man isn't free anymore. If we want to allow that more

then one woman can be engaged to one man then we can free the men if there are any women left. Doing that until every woman has been engaged results in a stable matching.

A problem with this approach is that it can result into a matching where all women are engaged with one man. This occurs if all women have the same man on the first place in their preference list. With the algorithm in its current form there is no possibility to prevent this.

4.1.3.4 Virtual entities

Another way to handle the situation doesn't involve any changes to the original algorithm but only changes to the input. For the members of the smaller set, virtual entities are created (including the associated preference lists) until the smaller set is of the same size as the larger set. That has to be done before the creation of the preference lists of the entities of the larger set because those have to include the virtual entities also. We created this approach ourselves and it appears unique for the stable matching problem. Later we saw similar approaches used in other systems, for example, in the Hungarian algorithm [Kuhn55].

In Figure 4.2 an example of virtual entities is shown. The set x consists of 4 entities and the set y of 8 entities. In Figure 4.2.**a** the virtual case is shown, here 4 virtual entities (denoted by character instead of digits) are added. In Figure 4.2.**b** is shown what the result is when the virtual entities are merged into the real ones.

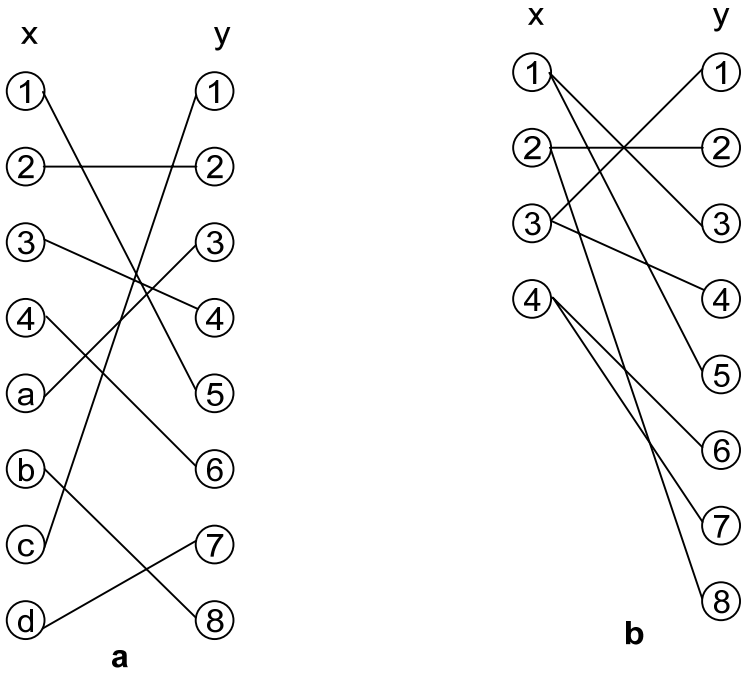

Figure 4.2: Virtual entities

4.1.4 Complexity of the stable matching algorithms

We will now discuss the complexity of the presented stable matching algorithms. As the used algorithm is an extension to the original version we will first discuss the complexity of the original.

4.1.4.1 Stable marriage

In every stable matching algorithm the defining action is the comparing of matchings. Is the proposed matching better or worse then the current matching? In the stable marriage algorithm for N men and N women we can easily determine that the maximum amount of possible matchings is the number of users time the number of subcarriers, so N^2. It isn't possible to say if all possible proposals are made without running the algorithm in advance, but it gives the worst case limit of the algorithm.

The N^2 possible proposals are only made in the worst case. In practice usually fewer proposals will be made before a stable matching is found. According to [Wils72] the average amount is $N \log_e N$. This was determined by running the algorithm many times for different sets of preference lists and different problem sizes.

4.1.4.2 Hospital/Resident

There are M hospitals and these all have a number of available places for residents. The total number of available places over all M hospitals we call M'. Now the number of possible "proposals" is the number of residents N times to total number of available places for residents per hospital M'. Therefore the worst case complexity is O(N*M').

4.1.4.3 Freeing entities

We now have M men and N women, the algorithm has been changed so all women have been matched once the algorithm terminates. The number of possible proposals still is the number of men times the number of women, but that is M*N now. So the complexity in worst case is O(M*N).

4.1.4.4 Virtual entities

Again we are dealing with M men and N women. Only in this case we add virtual men to the number of men until it is the same as the number of women. So we end up with a similar situation as with the original algorithm. Therefore the this version of the algorithm has a worse case complexity of $O(N^2)$.

4.1.5 Using the stable matching algorithm

When we say we use a stable matching algorithm in an assignment method we mean that we use an algorithm that results in a stable matching. Which version of algorithm exactly differs per situation and will be mentioned when it is used.

We stop calling the entities man and woman or hospital and residents and don't use the associated terms (e.g. proposal, engagement, etc.) anymore either. From now on, we use ONUs and subcarrier, which are assigned to each other, forming an assignment. We assume that there are more subcarriers then ONUs; to be exact, the number of subcarriers is an integer multiple of the number of ONUs.

To ensure some fairness between the ONUs we want to prevent starvation. This can be achieved in several ways. The simplest one is to assign every ONU the same integer number of subcarriers such that all subcarriers are assigned. It should be mentioned that this approach is not be the optimal one. Every ONU gets the same amount of subcarrier, but it might not need that many subcarriers or could use more subcarriers then the fixed amount.

Therefore another possible approach would be to dynamically distribute the subcarrier over the ONU based on the traffic demands of the ONUs. Such an approach will likely give a more optimal result, but is also more complicated to implement. So in this thesis we will only use the simple approach for determining how many subcarrier each ONU is assigned.

Using a fixed amount of subcarriers per ONU restricts us in choosing which algorithms we can use, only the virtual user and hospitals/residents versions are usable. Both versions result in the same stable matching, so on this issue it doesn't matter which one is used. Complexity wise there isn't a difference between both versions either. But on the issue of implementation the virtual user variant is the easier one as it
The preference lists are based on the normalized SNRs of the subcarriers for ONUs and on the queue lengths of the ONUs for the subcarriers. For more information on this and the actual implementation of the algorithms we refer to chapter 5 on simulation.

As mentioned before the virtual user case requires some extra work during the creation of preference lists. The virtual entities have to be created and later the preference lists have to reflect that.

4.2 Hungarian algorithm

The Hungarian algorithm is an algorithm that can be used to solve a so called assignment problem. In this section we first explain what exactly an assignment problem is. We then continue with explaining how the Hungarian algorithm is used to solve such an assignment problem. This section is ended with explaining how we used the Hungarian algorithm in a subcarrier assignment method.

4.2.1 An assignment problem

The assignment problem is a well known combinatorial optimization in the operational research. In its basic form it is usually presented as follows. There are a number of agents and tasks. Each agent can perform one task for a certain cost; this cost can differ per agent / task combination. Now we want to assign to each agent exactly one task and such that the total cost of the assignment is minimized. When there are an equal number of agents and tasks, then the sum of all agent costs is the total cost and we call it the linear assignment problem.

4.2.2 The Hungarian algorithm

The Hungarian algorithm is one of the algorithms which can be used to solve such a linear assignment problem in polynomial time, with a complexity of $O(N^3)$. Its first version, which was called the Hungarian method then, was published by Harold Kuhn in 1955 [Kuhn55]. This method was based on work by two Hungarian mathematics Dénes König and Jenő Egerváry, therefore the name Hungarian method. The Hungarian method was revised by James Munkres in 1957 [Munk57] and since then has been called the Hungarian algorithm (also known as the Munkres assignment algorithm or the Kuhn-Munkres algorithm).

For the Hungarian algorithm the assignment problem is modelled as an N x N matrix as shown to the right. In this matrix the elements $c_{i,j}$ are the cost for performing a certain task j by a certain worker i. So the workers are the rows and the tasks are the columns. This matrix is called the cost matrix. On this cost matrix the Hungarian algorithm finds the solution with the lowest total cost.

$$c_{i,j} = \begin{bmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,n} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,n} \\ & \ddots & & \\ c_{n,0} & c_{n,1} & \cdots & c_{n,n} \end{bmatrix}$$

The algorithm can be expressed in the following steps

1. subtract the lowest value in every row from each value in that row.
2. subtract the lowest value in every column from each values in that column.
3. try to cover all the zeroes with as little as possible lines (both vertical and horizontal). If the number of lines is the same as the size of the matrix find the solution (indicated by the zeros in the matrix). If the number of lines is smaller then the size of the matrix, find the smallest uncovered value. Subtract this value from each uncovered value and add it to all values at the intersections of the lines.
4. repeat step 3 until a solution is found.

Let's look at a simple example; we start with the following cost matrix.

$$\begin{bmatrix} 3 & 4 & 2 \\ 2 & 1 & 5 \\ 3 & 8 & 6 \end{bmatrix}$$

After subtracting 2 from the values in the first row, 1 from the values in the second row and 3 from the values in the third row, we get the following matrix.

$$\begin{bmatrix} 1 & 2 & 0 \\ 1 & 0 & 4 \\ 0 & 5 & 3 \end{bmatrix}$$

In this situation the algorithm is already done after step 1, step 2 isn't required anymore, this is of course not always the case. The matrix shows us that $c_{1,3}$, $c_{2,2}$ and $c_{3,1}$ form the minimized result. So worker 1 is assigned task 3, worker 2 is assigned task 2 and worker 3 is assigned task 1.

As mentioned before the Hungarian algorithm works on a square matrix. When this isn't the case, it can be handled for example by padding the smaller side with zeros such that the matrix eventually becomes square. In this way, it won't interfere with the algorithm.

4.2.3 Maximization

In some circumstances we don't want to minimize the problem but maximize it. This happens for example when the values in the matrix aren't costs, but profits. The Hungarian algorithm can handle this very simply. It is done by first determining the highest value and then subtracting every value from this and then running the Hungarian algorithm as normal.

Let's look again at the previous example, first we find the highest value, which is 8 in this case and subtract every value from that, this gives the following matrix.

$$\begin{bmatrix} 5 & 4 & 6 \\ 6 & 7 & 3 \\ 5 & 0 & 2 \end{bmatrix}$$

Subtracting the lowest value from each row from all the values in that row gives the following matrix.

$$\begin{bmatrix} 1 & 0 & 2 \\ 3 & 4 & 0 \\ 5 & 0 & 2 \end{bmatrix}$$

Then subtracting the lowest value from each column from all values in that column gives the following matrix

$$\begin{bmatrix} 0 & 0 & 2 \\ 2 & 4 & 0 \\ 4 & 0 & 2 \end{bmatrix}$$

In this matrix we see that $c_{1,1}$, $c_{2,3}$ and $c_{3,2}$ form the maximized solution.

4.2.4 Using the Hungarian algorithm

When the Hungarian algorithm is used for subcarrier assignment, the following approach is used. The matrix is of the profit variation containing the normalized SNR values of the subcarriers for the ONUs. As there are more subcarrier then ONUs, this would mean the matrix is not square. But like in the case of stable matching algorithm we create virtual ONUs such that, once again, we end up with having a square matrix.

The Hungarian algorithm optimizes the total profit. It doesn't care from which ONU the profits come, just that the sum of individual normalized SNRs is the highest possible. This means that in general it won't favour certain ONUs over others based on anything expect the possible profit.

5 Simulation description

In this chapter we will discuss the simulation that is used to test and compare the developed subcarrier assignment methods.

The first section is on the development of the simulation model in OPNET. We will give a theoretical and implementation overview of the system model, assignment methods and bit loading. After that a section on the inputs of the simulation follows. Then the simulation outputs and statistical analysis that are preformed on those discussed. Finally we present and discuss the results we obtained with the simulation.

5.1 Simulation development

In this section we describe the development of the OPNET simulation model. For all parts we first give a theoretical description and then proceed with details on the implementation.

5.1.1 The system model

We start the development section with the system model that is used for the simulations.

In the following sections we will sometimes discuss the location in the simulation where a certain task is performed. Some of these tasks might seem strange or unrelated to their location in a real system, but to prevent making the simulation too complicated they are performed as mentioned. For example a real system wouldn't make up impulse response figures at the OLT.

First we present an overview of the separate sections of the model in Figure 5.1.and what they globally do. Not everything will be directly clear, but that is all explained in the rest of this section.

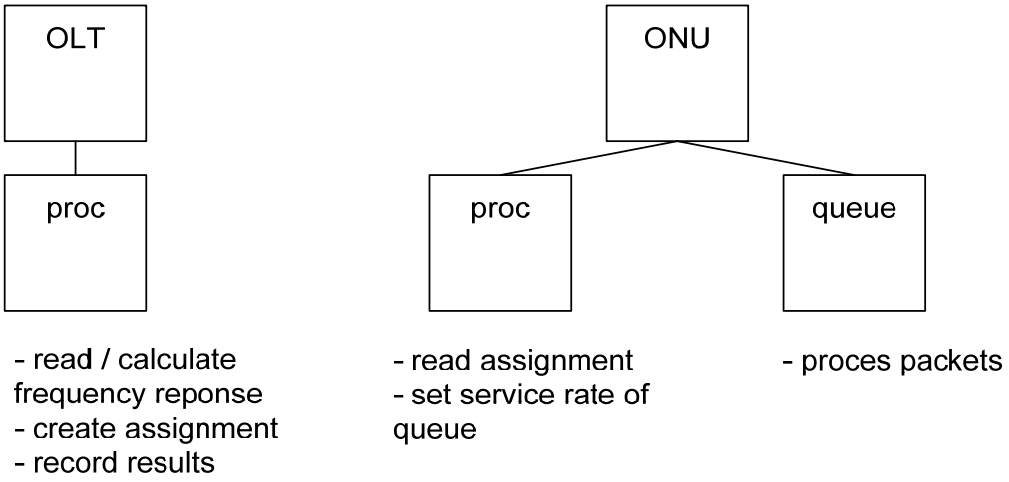

Figure 5.1: The system model

5.1.1.1 Theoretical model

Starting with the most basic parts of the model; it consists of 1 OLT, 16 ONUs and 128 subcarriers. The subcarriers are 10 MHz wide and the spacing between two subcarriers is 20 MHz, from the centre frequency of one to the centre frequency of the next. The first subcarrier starts around 5 GHz and with the mentioned spacing the last one starts around 7.54 GHz. In Figure 5.2 below the frequency response chart is shown.

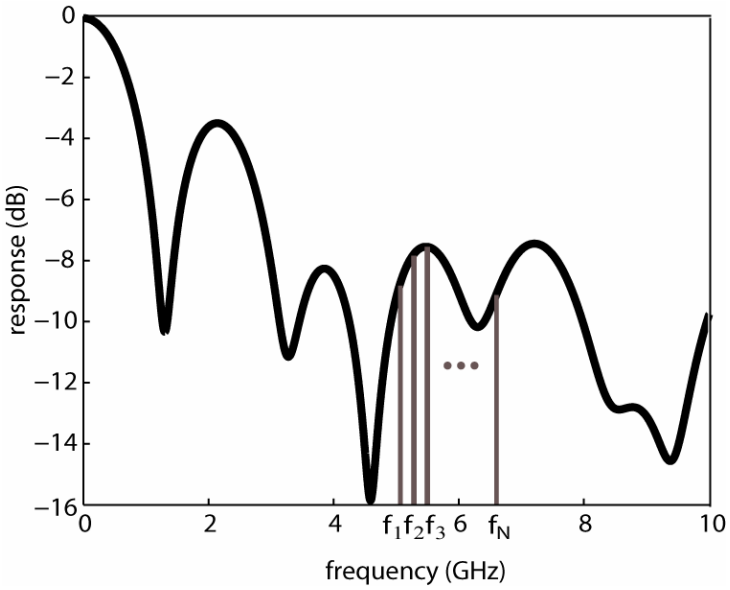

Figure 5.2: Frequency response

5.1.1.1.1 Timing

We decided on creating a new assignment every 100ms, because with this value every subcarrier should have an approximately flat frequency response. Such a period we will call the time epoch. We assume that during each such time epoch the link characteristics are constant, they just change after every time epoch. With these settings we believe we offer variable enough link characteristic in order to reasonably examine the different assignment methods.

In Figure 5.3 a timeline is shown of a typical execution of the simulation, we focus here on the first section up to 0.8 seconds.

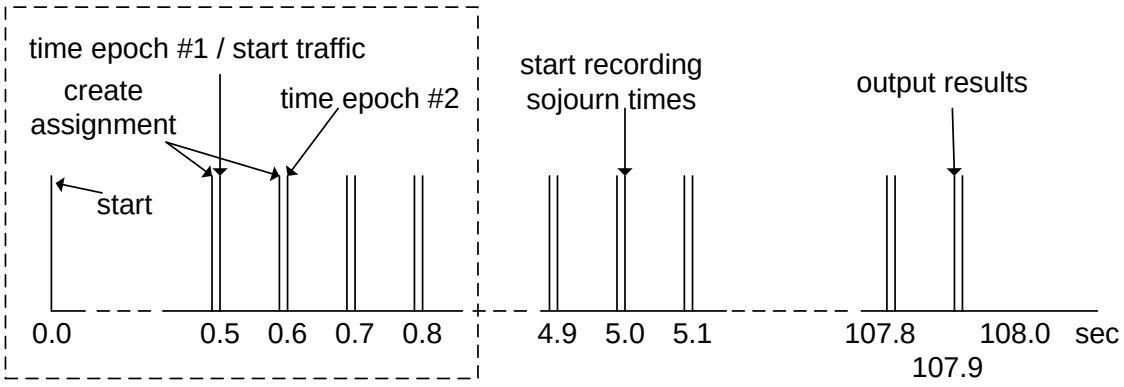

Figure 5.3: Simulation timeline, initialization

The simulation starts at 0 second. The first assignment is created a microsecond before the first time epoch which starts at 0.5 seconds. The traffic generation also starts at 0.5 seconds. This initialisation time of 0.5 seconds is used so all parts of the simulator have read the correct parameters once the subcarrier assignments creation starts. The exact value is somewhat arbitrary and partly a leftover from an earlier version of the model.

5.1.1.1.2 Link quality

The master assignment is about studying subcarrier assignment algorithms in a multimode PON where consequently the line quality is time varying. As we don't have measurements from a real system, a model was created to represent the link.

This link model is based on the calculation of impulse responses from the book "Principles of optical fibre communication" by van Etten [EtPl00]. First the number of available modes in a certain strand of fibre is determined. This number is always the same for a certain set of parameters and for the ones used here the number of modes is 36. The parameters for this fibre are: wave length of 1310nm (the best value for a low chromatic dispersion in a multimode silica fibre), fibre with parabolic refractive index profile, fibre core radius 25e-6, numerical aperture of 0.2, refractive index core 1.5, refractive index cladding 1.4866. Then the specific group delays for these modes are

calculated, finally those are multiplied with the length of the fibre, to create group delays τ_g. For this set of group delays we determine the lowest and highest value of group delays to define a group delay interval.

Let's start with the expression for the impulse response, it is shown in (1).

$$h(t) = e^{-2\alpha z} \frac{1}{M} \frac{R}{K} \sum_{i=1}^{K} \delta(t - \tau_{g(i)} z) \tag{1}$$

The variables used are *t* for the time, α for the attenuation coefficient (in Neper/m), *z* for the distance (in meter), *R* for the responsivity, *K* for the number of available modes, *M* for the number of ONUs, δ for the delta function and $\tau_{g(i)}$ the group delay for mode *i*. Function (2) then shows the calculation of the frequency response for a certain frequency *f*.

$$H(f) = e^{-2\alpha z} \frac{1}{M} \frac{R}{K} \sum_{i=1}^{K} e^{-j2\pi f \tau_{g(i)}} \tag{2}$$

The variables are almost all the same; α for the attenuation coefficient (in Neper/m), *z* for distance (in meter), *R* for responsivity, *K* for the number of available modes and *M* for the number of ONUs.

Because we want to simulate a time varying link the following approach was used. For a certain number of modes K, we can generate K uniformly distributed random numbers in every time epoch. These numbers are then scaled and shifted in previously mentioned group delay interval and with each different realization of those random numbers another instance of the responses can be calculated.

For the simulation, we further determine the normalized SNR of every channel for each ONU which is shown in expression (3).

$$\gamma(f) = \frac{|H(f)|^2}{N(f)} \tag{3}$$

In the expression $\gamma(f)$ is the normalized SNR, $|H(f)|$ is the frequency response magnitude and $N(f)$ is the power spectral density of the noise. Regarding the noise, only the thermal noise is considered here, as we assume a thermal noise limited system, via the expression (4) below.

$$N(f) = \frac{2k_b T}{R_l} = \frac{N_0}{2} \tag{4}$$

The variables used here are; k_b for Boltzmann-constant (in Joule/Kelvin), *T* for temperature (in Kelvin) and R_l for the load resistance (in Ω). Because the expression is

independent of the frequency the noise level is a constant over the frequency range of interest, which will be noted as N_0.

The frequencies that are used for the calculation of the normalized SNR are those of the 128 subcarriers. The distances, which are associated with the ONUs, are predetermined at the start and don't change during a simulation.

Essentially, we have a matrix containing normalized SNR values for all ONUs on all subcarriers for a certain time epoch. By doing these calculations for every time epoch there will be new normalized SNR values each new time epoch. This way we simulate the varying link qualities in which the subcarrier assignment methods will be applied.

5.1.1.1.3 Queue

The final part of the system model are the queues, they represent the data that the ONUs or OLT wants to transmit over the network. As any normal queue they have an arrival rate and service rate. During a simulation the arrival rate will be exponential but with a fixed mean unless otherwise stated. The service rate will be dependant on the number of bits per symbol that have been loaded on the subcarriers assigned to an ONU.

5.1.1.2 Model implementation

We will now discuss how the theoretical model was implemented in the simulation environment OPNET. For more specific info on the implementation in OPNET we point to appendix A. Starting at the network level, one OLT node and 16 ONU nodes were created. The subcarriers aren't modelled as visible objects in the simulator.

5.1.1.2.1 Timing

In OPNET timing is done via interrupts, at the OLT we schedule an interrupt to trigger a microsecond before the starting time of 0.5 seconds. When that interrupt triggers, it will cause the assign() function to be called. That function generates the normalized SNR values, determines the subcarrier assignment and finally schedules another interrupt to happen in the next 100ms. This way we create a subcarrier assignment just before the start of every time epoch.
At the ONUs the first assignment is read at 0.5 seconds with the getSubcarrierAssignment() function. At the end of this function an interrupt is also scheduled to trigger 100ms later, so the next assignment is read at the start of the next time epoch.

5.1.1.2.2 Link quality

The implementation for the link quality is based on a implementation of the fibre model in MATLAB. That code was translated in C and changed a little so it could be used in OPNET. The code containing functions for calculating the link frequency response and noise level is located in the separate file snrCalculation.cpp. The related functions are called from the proc processor module in the OLT by calling the previously mentioned assign() function. Just before a time epoch starts, the OLT becomes active and starts creating a normalized SNR matrix for the upcoming time epoch via those functions for the concerned distances and frequencies.

The written code is mostly a one on one mapping of the appropriate formulas into computer code. Though one change was made here during the implementation, we now use a scaling factor of 2 is applied to the specific group delays due to the non-ideality of the components of the system (i.e. the coupler).

Both the link frequency responses (and therefore the normalized SNR values) and the distances can be stored after they have been generated. In this way, different assignment methods can be simulated on the exact same data.

5.1.1.2.3 Queue

The queues are implemented with a simple queuing system in the ONUs, every ONU has one such system. As can be seen in Figure 5.4 this system consists of a source, queue and sink processor.

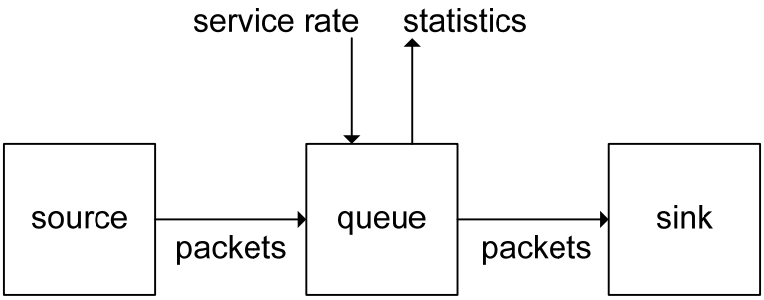

Figure 5.4: A simple queueing system

The source has as parameters the packet interarrival time and packet size, which both have a certain chosen distribution set via the simulator. The queue only has a service rate as parameter. During the simulation the source creates packets according to its parameters and sends those instantly to the queue. There packets are processed based on the service rate. Finally the packets are send to the sink where they are destroyed. The only parameter that changes in the queuing system during the simulation is the service rate. It will be set, every time epoch, based on the number of bits loaded on the subcarriers assigned to a certain ONU.

Formally this system would only model the behaviour during uplink traffic as the queues are located at the ONUs. But during one simulation run only uplink or downlink traffic is simulated the same queue can then be used for downlink traffic. In that case they represent data the OLT wants to send to the ONUs, with data for every ONU in a different queue.

The service rate for the queue is set via a statistic wire (for more information see appendix A) from the proc processor module. At the start of each time epoch the ONU checks in the proc for its current subcarrier assignment and loaded bits and from that calculates the service rate for that epoch. That value is then set on the statistic wire and will be used by the queue.

We use the queues to simulate the traffic ONUs want to send or receive; the amount of traffic is measured via the queue size. The larger the queue size the more traffic that needs to be transmitted. The queue size is determined inside the queue processor module, every time a packet arrives the queue size is updated. Via a globally shared array containing the queue sizes of all ONUs the current values can be read instantaniously when required by the OLT.

5.1.2 Assignment methods

We continue the development section with discussing the used assignment methods in the simulation. Starting with theoretical descriptions and following with the implementation details. For some of the methods a more extensive theoretical background already has been presented in the previous chapter 4, so here we will mainly focus on the parts important for the simulation.

5.1.2.1 Theoretical

5.1.2.1.1 An assignment

Before we introduce the assignment methods lets look at an assignment, a created assignment consists of 128 (user, subcarrier) pairs, in which all subcarriers appear once. How many times each user appears in the assignment depends on the assignment method. Each subcarrier assignment method has to result in such an assignment.

5.1.2.1.2 Stable matching (virtual users)

Before the stable matching algorithm can be performed preference lists have to be created. This is done based on the subcarrier quality for the users and on the queue size of the users for the subcarriers. In the virtual user case some extra work is required so all

virtual users have preferences lists and are all added on the right location in the preference lists of the subcarriers.

When the preference lists are created then the virtual user version of the stable matching algorithm is executed as described in the previous chapter 4. Once the algorithm is done it has produced a subcarrier assignment.

With this subcarrier assignment as input the bit loading algorithm will be executed to determine how many bits to load on each subcarrier. The details on the bit loading can be found in section 5.1.3.

5.1.2.1.3 Contiguous

Here the subcarriers are assigned in contiguous blocks of 8 to each user. Starting with subcarrier 1 through 8 to user 1, subcarrier 9 through 18 to user 2 and so on. This results in a simple subcarrier assignment on which bit loading can be performed. A contiguous assignment doesn't change during the simulation, still bit loading happens every time epoch, therefore we call it semi-static.

5.1.2.1.4 Interleaved

For the interleaved method subcarriers are also semi-statically assigned. Now a user is assigned 8 subcarriers evenly spread out over the available ones. User 1 is assigned subcarrier 1, 17, 33, 49, 65, 81, 97 and 113, user 2 is assigned subcarrier 2, 18, 34 and so on.

5.1.2.1.5 Hungarian

Let's look at how the Hungarian algorithm is applied to our case; we have the normalized SNR values for all 16 users on all 128 subcarriers and want to find the best possible overall throughput. This means that we want to find the highest normalized SNR values, which can be seen as the profits here. So we have to use maximization and have to copy the normalized SNR value lines 7 times so the profit matrix is equal sided. After the algorithm has determined the maximum solution we just have to identify to which actual users the virtual users belong and we have a subcarrier assignment ready for bit loading.

5.1.2.2 Implementation

We will now discuss the implementations of the assignment methods in OPNET, starting with some global remarks about the programming. All the assignment methods are located in a separate C file called assignmentMethods.c. This was done so they could be

tested easily outside the OPNET environment. Initially the methods were spread over several files, but that required changing #include lines in the OPNET code and recompiling every time another algorithm was used so it was changed to just one file.

The C file is included in the OLT proc processor header section and during the assignment the selected start method function is called with the appropriate parameters. These parameters include the number of users, number of subcarriers, the SNR values, queue size (for stable matching), impulse response values, the subcarrier assignment in which store the calculated subcarrier assignment, the traffic direction and some debug parameters. Beyond calling that start function very little of the subcarrier assignment methods code is actually in OPNET. Once the start function has finished a complete subcarrier assignment including loaded bits is available in the OLT. The OLT then has to make it available for the ONUs so they can use that assignment for the coming time epoch.

5.1.2.2.1 An assignment

In the simulation the assignment will be a two dimensional array containing for each assignment the subcarrier, the user and the number of loaded bits. After an assignment method has been executed the number of loaded bits will be unset until the bit loading algorithm has been executed.

5.1.2.2.2 Subcarrier assignment method

Before looking at specific assignment methods we will first describe the general approach used. Because this is partly identical for the different methods we prevent explaining the same steps over and over.

All or some of the following tasks are performed in the start function of an assignment method:

- Transform input from OPNET into usable values
- Perform assignment
- Transform result into generic form
- Perform bit loading
- Transform results into output to OPNET

The first task is only performed for assignment methods that determine the subcarrier assignment based on certain input. From OPNET the SNRs and queue sizes are passed on and in this task those are transformed into the required input values.

The assignment itself is always performed, for some methods this is very simple and for others it requires some more work.

We started with several different implementations to the assignment methods, all using there own data structures. Also initially not all implementation included code for bit loading. Once we took one implementation of the bit loading and wanted to use that for other subcarrier assignment implementations we had to change the data structures between the steps so it was possible to use the same bit loading code everywhere.

As the bit loading is pretty related to the assignment we decided to perform it here also. In this way it was also easy to test the bit loading functions outside of OPNET. The bit loading is identical for all different methods and is performed for all of them.

Because we defined a certain subcarrier assignment structure in OPNET the results of the previous tasks need to be transformed into that structure at the end of the start function.

5.1.2.2.3 Stable matching (virtual users)

The implementation of the start function of the virtual user version of the stable matching assignment method begins with transforming the SNRs values and the queue size into preference lists which are used as input for the stable matching algorithm.

Initially the creation of these preference lists was done via simple sorting. The highest SNR was put first in the preference list. But while looking at the data, especially with the queue size we noticed that there were often equal queue sizes. This makes sense as under light load the queues will often be empty or contain a single packet. When we encountered such a group of equal values their order in de preference list was determined based on the order of the queue sizes from OPNET. That order was on ONU number, so with equal sizes the ONUs were put into the preference list based on their number. This clearly favours lower numbered ONUs over higher numbered ONUs and can cause unwanted effects on the subcarrier assignment and bit loading. We have solved this issue by randomizing the order of the users or subcarriers with the same values before they are put in the actual preference list.

At the beginning of the assignment part of the method some extra work is required to create the virtual users and updated the preference lists of the subcarrier to include these virtual users. When all virtual users are created and the preference lists are filled then the stable matching algorithm can be executed. The implementation of that is basically a one on one mapping of the pseudo code shown in section [x] into c code. Once the stable matching algorithm is finished the virtual users have to be merged into their actual versions and the subcarrier assignment is done.

Now some transforming of values is required in preparation of the bit loading, then bit loading is performed and after that the subcarrier assignment is transformed in its final form. As the bit loading implementation part is identical for all methods it is explained once in section 5.1.3.

5.1.2.2.4 Contiguous

The contiguous assignment method uses the start function of stable matching method. But as there is no special input required that task is skipped here. And instead of a complicated algorithm there is just one loop assigning the subcarriers in contiguous blocks of 8 to the users. Then the usual code involving the bit loading is executed.

5.1.2.2.5 Interleaved

The implementation of the interleaved method closely follows the contiguous one. But now the subcarriers are assigned according to the interleaved method described earlier.

5.1.2.2.6 Hungarian

The implementation of the Hungarian algorithm itself is done with code from Cyrill Stachniss [Cyri04], which it is an enhancement of the version from the Stanford GraphBase [Knut93].

The Stanford GraphBase is a set of functions and datasets that can be used to generate and work with graphs and networks. It is developed by Donald Knuth at the Stanford University. One of the functions available in the package is for performing the Hungarian algorithm.

This function was written in C but targeted at a special case of a certain matrix. Brian P. Gerkey [Bria04] took the code from the Stanford GraphBase and based on that created a more general implementation. This implementation suffered from a deadlock problem in certain obscure cases which the author was unable to fix. Eventually they were fixed by Cyrill Stachniss who released a new version himself.

Because this code comes with its own input format and result format some more work was required to transform everything in correct external formats.

Also because of this the implementation first started in a separate c file and was later on merged in the assignmentMethod.c file. It still has its own start function in which the first task is transforming the SNR values from OPNET into a cost matrix. Once this is done the hungarian_solve function from the code by Stachniss is called to determine to optimal solution.

Then, as for the other methods, code is executed to transform the previous solution into a subcarrier assignment on which bit loading can be performed. Again bit loading is performed and the total subcarrier assignment is created to be returned to OPNET.

5.1.3 Bit loading

This last part of the simulation development section will be on bit loading. Bit loading is the system where is determined how many bits are loaded (available) per symbol on every channel. This can be done statically, by just setting the number of bits per symbol to a certain value. But arguably a better way is to let the bit loading be dependent on the channel quality characterized by the normalized SNRs. This dynamic approach is what is used in this subcarrier assignment scheme.

In our system the bit loading follows the subcarrier assignment itself, first it is determined which users get which channels. Then for every channel it is determined how many bits are loaded.

5.1.3.1 Theoretical

We start the description of the theoretical bit loading model with the received signal at a certain ONU *j* at a certain channel *i* which is shown in (6). In this expression P_0 is the optical power, $H_j(f_i)$ is the frequency response of a certain user at a certain frequency, m_i is the optical modulation index and $s_i(t)$ is the modulating signal.

$$\tilde{s}_{i,j}(t) = P_0 \mid H_j(f_i) \mid m_i s_i(t) \qquad (6)$$

We will be using QAM modulation with an even number of bits per symbol, a square constellation and a maximum of 8 bits per symbol. The required minimum distance between two constellation points for a certain BER can be formulated as shown in (7). $N_0/2$ is the noise PSD for which we assume only thermal noise is a factor here. The expression we use for the noise level is $2k_BT / R_L$, where k_B is the Boltzmann constant of 1.3806505×10^{-23} in Joule/Kelvin, T the temperature of 300 in Kelvin and R_L the load resistance at the receiver which is 50Ω. The Q^{-1} is the inverse Q function and P_e is the BER which will be 10^{-9} in the simulation.

$$d = \sqrt{2N_0}\,Q^{-1}\left(\frac{P_e}{4}\right) \qquad (7)$$

We now give the relation between that distance and the required signal amplitude (per symbol) of a certain subcarrier. For this we express $s_i(t) = a_i \cos(2\pi f_i t + \varphi_i)$ and also $\tilde{s}_{i,j}(t) = \tilde{a}_{i,j} \cos(2\pi f_i t + \varphi_i)$. With those expressions during one symbol time $T_s = \frac{n}{f_i}$, where n is a positive integer, so the symbol time is integer multiple of the frequency, d is related to $\tilde{\tilde{a}}_{i,j}$ as shown in (8), for the outer most constellation points.

$$\widetilde{\widehat{a}}_{i,j} = \frac{d}{\sqrt{T_s}}\left(2^{\frac{c_i}{2}} - 1\right) \tag{8}$$

In this expression c_i is the number of bits we want to load. This implies the following.

$$m_i\widehat{a}_i = \frac{d}{P_0 \mid H_j(f_i) \mid \sqrt{T_s}}\left(2^{\frac{c_i}{2}} - 1\right) \tag{9}$$

For now we want to avoid clipping, so the following expression (10) describes the resulting constraint which on the downlink transmission.

$$\sum_{i=1}^{N} F_{i,j}(c_i) \le 1 \tag{10}$$

$$F_{i,j}(c_i) \stackrel{\Delta}{=} \frac{d}{P_0 \mid H_j(f_i) \mid \sqrt{T_s}}\left(2^{\frac{c_i}{2}} - 1\right) \tag{11}$$

In this summation every term can be seen as the cost of loading c_i bits on subcarrier i belonging to user j and the total cost has to satisfy (10) to be allowable. Following this expression we can define an expression for calculating the cost of the next step as shown in (12). With the next step we mean the cost to add an additional 2 bits while the concerned subcarrier has already been loaded with c_i bits.

$$\Delta F_{i,j}(c_i) \stackrel{\Delta}{=} F_{i,j}(c_i + 2) - F_{i,j}(c_i) \tag{12}$$

With this expression we can use a greedy approach by every time selecting the subcarrier with the cheapest step additional cost. This is followed by calculating the value of the accumulated cost for that chosen subcarrier and the cost of the next step. By keep doing this while the constraint in (10) is still satisfied, we get the optimal bit loading.

5.1.3.2 Implementation

The first version of the bit loading came from the multiple access method [xx] subcarrier assignment algorithm. There it is the final step and based on a fixed amount of available power for all users and looks at the channel SNR. The pseudo code for that algorithm was turned into an implementation. Once we wanted to add bit loading to the stable matching and later on the Hungarian method we used that same bit loading implementation from the multiple access method.

As the initial version of the bit loading wasn't targeted to an optical system some changes were required to get to the implementation based on the above theoretical system. Still we were able to keep large parts of the previous implementation but now the total cost was the main limiting factor where it was the power before.

The implementation of the bit loading itself is rather simple. It starts by calculating the costs for loading 2 bits for every channel. Then in a while loop it will select the subcarrier of which the cost of loading 2 bits is the lowest. After that the cost of loading another 2 bits on that channel is calculated. Now the system will look again for the lowest cost to load 2 more bits and update the cost for the selected channel. Every time a bit is loaded the cost of that is stored in the current total cost variable. Once the cost of loading 2 more bits plus the current total cost becomes larger then the set limit of 1.0 then the while loop ends and the bit loading stops.

During the loop we kept track of how many times bits where loaded on which channels and that information can be used in a final transforming data step to create a complete subcarrier assignment.

The cost function itself isn't that simple as can be seen from the theoretical section. The implementation of it wasn't that complicated though, just mapping the mathematical functions into computer code and calling them in the correct order.

5.1.3.2.1 Uplink / Downlink

In the initial bit loading implementation and also several of its revisions bit loading was performed for an uplink situation, so traffic going from the ONUs to the OLT. In this case every ONU has a certain amount of power it uses for bit loading on the subcarriers that have been assigned to it. That process is then repeated for all users to get a complete bit loading.

Because we also wanted to simulate downlink traffic we created another implementation for that. Now the OLT has a certain amount of power and bit loading is performed for all subcarriers at once instead of the ones assigned to one user.

5.1.3.2.2 The transforming code

As mentioned the implementation of the bit loading came from a certain larger piece of code for the multiple access method. This means that it continues using certain data structures that were started in the first section of the implementation. Once we just took this implementation and used it in other assignment methods the data structures didn't match. At that time we decided to just transform the required data before the bit loading into the structure the bit loading can use. And after the bit loading it is transformed to a form that can be used by OPNET. When the bit loading changed internally some small

changes were required on the transforming functions but the basic system remained in place.

5.2 Simulation input

In this section we describe the inputs for the simulation. With the inputs we mean the data that is used during the simulation and the settings that control how the simulation is running.

5.2.1 Input data

This is the data the simulation uses as a primary input, it describes the simulation behaviour.

5.2.1.1 Distances, seeds, responses

The main input data are the frequency responses; they are what determines the channel SNR values and they are used in the bit loading cost function. They are dependent on the distance between the ONU and OLT and the random chosen values.

5.2.1.2 Traffic

The other input data is the traffic and then exact flow of the traffic. Of course the type and mean of the distribution are known but the exact instance of it is determined by the OPNET seed value.

5.2.2 Settings

We can set a whole range of parameters for every simulation. This can be done via OPNET simulation, global or local (object) variables, per node in the network model or by setting them in the programming code. Here follows a list with descriptions of them.

Duration
This is the amount of simulation time, not clock time, the simulation will be running, it is an OPNET simulation variable.

Seed
This is the seed value for the random function used by OPNET in for example distributions used in traffic generation. It can be set to vary of multiple values for different runs and is an OPNET simulation variable.

Number of ONUs
The number of ONU nodes in the system, this always is 16 in the simulations. This parameter is set via an OPNET global variable.

Number of channels
The number of available channels in the system, this value is set to 129 because it used to include space for a control channel. This control channel isn't used anymore in the current simulation and everywhere this value is of importance 1 is subtracted to have 128 channels. This parameter is set via an OPNET global variable.

Distance source
With this parameter that is set via an object variable in OPNET we control the source of the distances between the ONUs and OLT. They can be either generated as described in section [x] and then be written to a file or be read from a file. If we generate and store the values once we can then use them in the next simulation run to have an identical situation.

Response source
Similar to the distance source but now for determining the response values.

Seed source
Similar to the distance source but for the determining the seed values used in generating the response and in breaking ties in stable matching.

Assignment method
This parameter controls which assignment method to use during the simulation, it is set as an object variable in OPNET.

Scenario
This OPNET object variable was used to write the scenario number in the result output files, currently we use the OPNET network model simulation variable for that.

Traffic direction
With this parameter we control if downlink or uplink traffic is simulated, it is set as an OPNET object variable.

Data set number
This parameter controls which data set to use during the simulation. A data set consists of a related distances, responses and seeds file.

Traffic loads / interarrival times
The traffic load is the amount of data the source inside an ONU produces. This value can be set in the network model for each ONU independently via the promoted node attribute "Packet Interarrival Time".

Start measuring time

The time at which recording of a certain value starts, the value is set in the code. This value is used to handle for example the initial transient removal which is discussed in the next section.

Output data time
This is the time when values(s) we are observing are stored in a file or shown on the screen. This for example used to write after a certain number of time epochs has passed without exactly determining the simulation duration.

5.2.3 Cases

The above list shows us there are many settings and even more combinations of them. We could try to simulate all possible combinations but this would take too much time and result in a large amount of mostly useless data. To be able to get usable results we define a set of parameters which we will call a case.

In one such a case all parameters are fixed and if we then run all assignment methods they will be ran under identical circumstances. We have a number of these cases that differ in a number of the parameters, the unlisted parameters are identical. We define these cases so that the relevant measurements can clearly be observed but also try to keep the simulation environment as realistic as possible.

Case 1
Distances: uniformly distributed between 1000 and 3000 meters
Traffic loads: all 100 Mbps

Case 2
Distances: uniformly distributed between 1000 and 3000 meters
Traffic loads: alternating between 150 Mbps and 50 Mbps

Case 3
Distances: all 2000 meters
Traffic loads: all 100 Mbps

Case 4
Distances: all 2000 meters
Traffic loads: alternating between 150 Mbps and 50 Mbps

Case 5
Distances: all 2000 meters
Traffic loads: first 4 ONUs 150 Mbps and 50 Mbps for the other ONUs

Case 6
Distances: all 2000 meters
Traffic loads: first ONU 150 Mbps and 50 Mbps for the other ONUs

Case 7
Distances: all 2000 meters
Traffic loads: first ONU 200 Mbps and 50 Mbps for the other ONUs

Case 8
Distances: all 2000 meters
Traffic loads: first ONU 250 Mbps and 50 Mbps for the other ONUs

Case 9
Distances: uniformly distributed between 1000 and 3000 meters
Traffic loads: 150 Mbps on all ONUs

Case 10
Distances: uniformly distributed between 1000 and 3000 meters
Traffic loads: 150 Mbps on all ONUs
Time epochs: 576

Case 11
Distances: uniformly distributed between 1000 and 3000 meters
Traffic loads: 175 Mbps on all ONUs
Time epochs: 576

Case 12
Distances: uniformly distributed between 1000 and 3000 meters
Traffic loads: 200 Mbps on all ONUs
Time epochs: 576

5.3 Simulation outputs

To get results we have to define the simulation outputs. As can be seen in appendix A OPNET offers possibilities for this via probes. So far we haven't used those but the outputs by writing the values we are interested in to the screen or to files. These outputs can be "raw" values like the current queue size in bits or ones which we already performed some processing on during the simulation, for example the mean packet sojourn time.

First we present the timeline from a typical execution of the simulation in Figure 5.6, we have seen this figure before but now we focus on the last part.

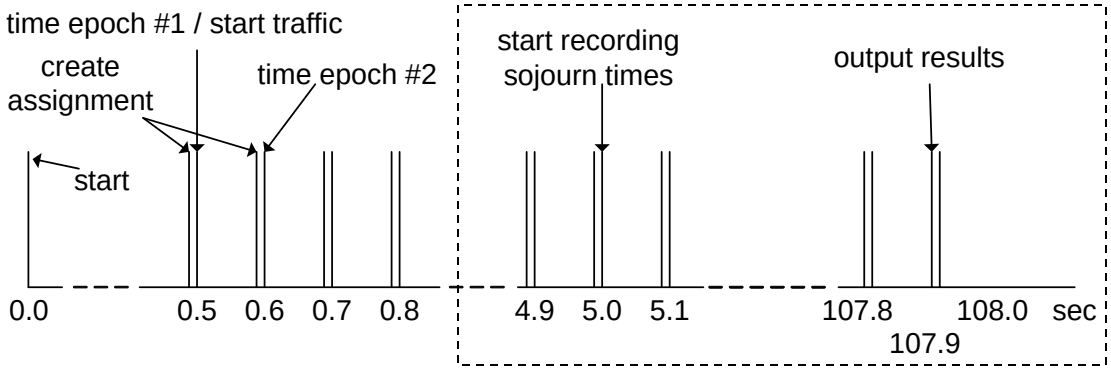

Figure 5.5: Simulation timeline

The writing of values to files might cause some delays due to the large number of values that needs to be written. But as the simulation is event driven it doesn't matter if an event takes longer in clock time.

Several values have been measured during the different stages of the simulation development. We will list the important ones now.

5.3.1 Variables for debugging

During the development of the simulation many values where measured to verify the correct behavior of the simulation. These were usually outputted to the screen so they could be checked easy and fast.

5.3.2 Assigned subcarriers

We measure the number of subcarriers that get assigned to each ONU during the whole simulation. This is done by counting them after every time the subcarrier assignment method has been executed.

5.3.3 Loaded bits per symbol

The number of loaded bits per symbol is determined every time epoch after the bit loading is finished. Then the total amount of bits loaded on the subcarrier assigned to one ONU is stored. Also the distribution of these total amounts is stored, so it is possible to see how often a certain total amount of bits was assigned per ONU.

5.3.4 Delays

Determining delays took a little longer then we expected at first. Initially we used a statistic wire from the proc node in the ONU to the queue node. But it was unclear what kind of delay was reported there and if it was an average or connected to a specific packet.

So to be sure we were measuring the correct value we went inside the queue node and used the op_q_stat() function with OPC_QSTAT_DELAY as argument to determine the delay at the end of every service completion. This delay was documented better so we know exactly what it consists off, it is the time a packet spends in the queue starting with its arrival and ending once it has been processed. This delay is also known as the sojourn time and that term we will be using in the future.

After every packet that is done processing we get a new sojourn time. We can record every instance of this value or add it to a total sojourn time variable. With such a total value variable and a sojourn time counter we can determine an average over the whole simulation run.

5.3.5 Statistical analysis

Because of the nature of simulation it is required to perform some statistical analysis on the obtained data. This section describes what we have done in that area.

5.3.5.1 Initial transient removal

Ideally we want to observe the simulation during a steady state situation, but when a simulation starts it usually isn't in such a state yet. That start up time in which the system is unstable is called the initial transient period. To get clear results we don't want to include that initial transient in our measurement of the sojourn times. There are several ways to remove this initial transient or to negate its effect. None of these methods are perfect as it is very hard to exactly determine when the initial transient period is over.

One way is by running the simulation long enough so the effect of the initial transient is neglectable on the total result. Another way is based on estimating the variance. All samples will be divided in batches with a certain batch size. We calculate the mean for all these batches and the variance of those batch means. Starting with a batch size of 2 and increasing it will cause the values of the initial transient period to get in the first batches. The variance will go down during the process because more and more values of the initial transient get into the first batch which will be the only big difference with the other batches. The batch size from when the variance starts to decrease monotonously is the number of samples that belong to the initial transient and should be removed from the total number of samples.

We recorded all the sojourn times for one simulation run and applied the above method. From those variances we determined that the system settled down starting at around 5 seconds. And as can be seen in the timeline in figure [x5] that is the moment when the measuring of the sojourn times starts.

5.3.5.2 Independent samples

Because a certain simulation runs with a fixed set of parameters and input values can results in non typical results we run them several times while varying one input value. Those results can be considered independent enough to draw conclusion from. We can just determine the average over those values but better is to also determine a confidence interval. That is a statistical method with which can we determine with a certain confidence (for example 95%) that the result lies in a certain interval.

5.4 Simulation results

After discussing the simulation development, its inputs and the outputs we finally get to the results now. At the start of every result a small overview of the used settings and the measurement we are interested in will be given, then the results are and a discussion of them is presented.

In general subcarrier assignment method the first possible result of interest is the number of assigned subcarrier to each ONU. But because with the used subcarrier assignment methods here that number of subcarriers per ONU is always 8 there is no need to give any attention to it. The next useful result involves the number of loaded bits.

5.4.1 Loaded bits per symbol

Within the measurement of the loaded bits there are several possibilities of what we exactly mean by loaded bits. The first thing we will look at is the average total amount of loaded bits per ONU per symbol.

5.4.1.1 Average loaded bits per symbol

For every time epoch an assignment is created and every ONU will end up with a number of bits per symbol it can use to transmit / receive data during that time epoch. That amount of bits we call the total loaded bits, we track this value during the simulation and determine an average of it in the end. This results in the average total amount of loaded bits per symbol per ONU.

For every case the simulation is executed for 5 different datasets and 15 different OPNET seeds for a total of 75 simulation runs for one subcarrier assignment method. Every dataset contains a different instance of the channel frequency responses, with a different seed for every time epoch. And every different OPNET seed causes a different drawing of random numbers for the traffic generation. The results of these 75 separate runs are then averaged to come to the final result which is presented here. This is done so that we are relatively sure the outcome wasn't caused by a specific combination of inputs. As it is possible that under certain circumstances a rare event occurs in which the results aren't representative at all for the system.

The first four cases are defined such to cover a large portion of situations of interest. Based on those cases and the results they provide other cases are used to further explore certain aspects of the algorithms. Therefore we first now present the results from those four cases.

5.4.1.1.1 Case 1

In this case the distances between the OLT and ONUs are uniformly distributed between 1000 and 3000 meters. After the first run we obtain the randomly chosen distances, for ONU 0 through ONU 15 they are: 2520, 2617, 2058, 1369, 1052, 2601, 2831, 2506, 2387, 1015, 2346, 1632, 2175, 1632, 2136, and 2590 meters. The traffic loads are set to 100Mbps for all ONUs. We simulate downlink traffic here and in all other cases unless mentioned otherwise. With these settings we execute the simulation once for all 4 assignment methods for 1076 time epochs. The result after the necessary statistical processing can be seen in Figure 5.6. In Figure 5.7 the previously mentioned distances between the OLT and ONUs be seen, also the "reverse distance" is visible, that is 3000 meter minus the actual distance.

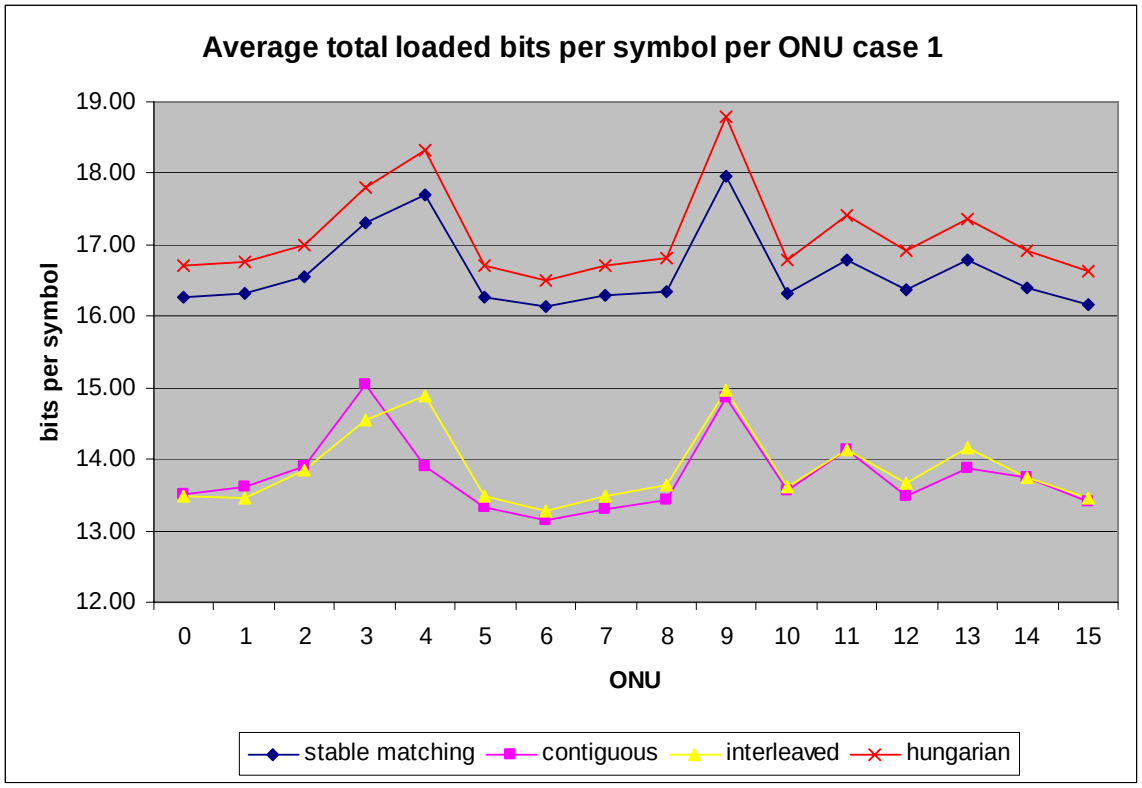

Figure 5.6

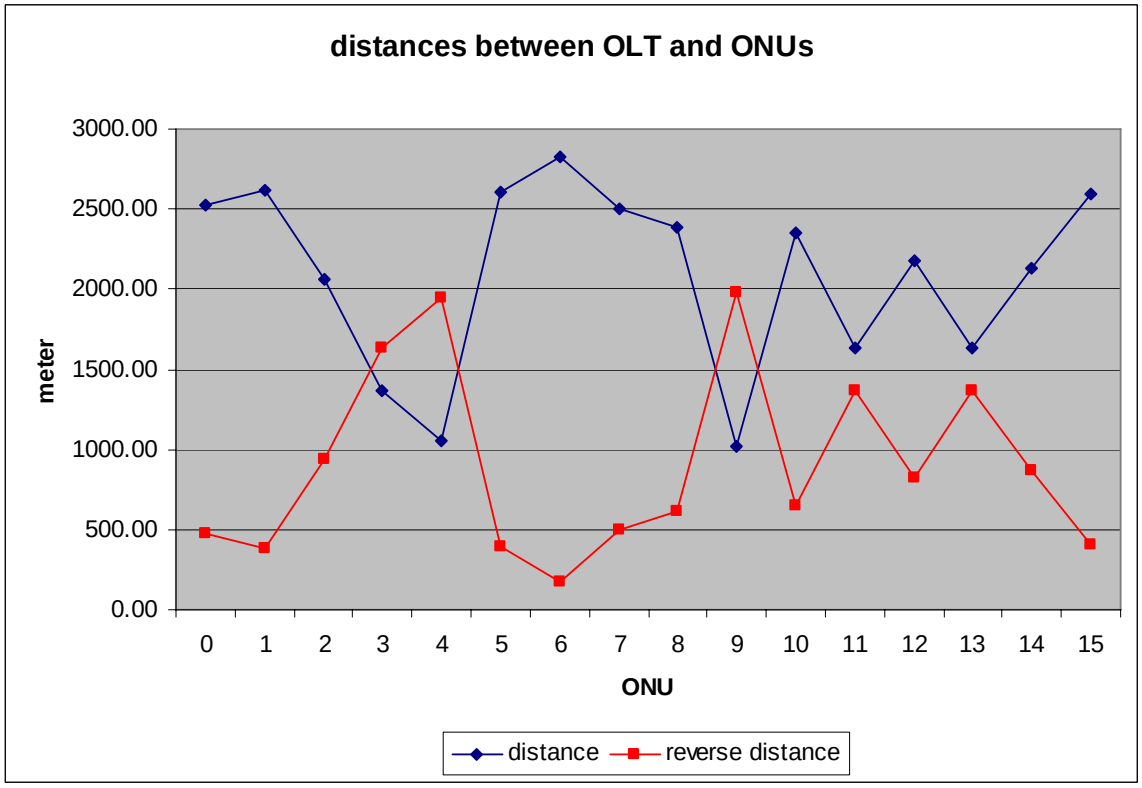

Figure 5.7

In the chart the dots on the lines are the actual data points, the connecting lines are used to make the behavior clearer. For the exact values see appendix B. The values on the y-axis are the amount of bits per symbol and the values on the x-axis are the ONU numbers. The chart shows that the Hungarian method is results in the most loaded bits, closely followed by the stable matching method. The contiguous and interleaved methods clearly get assigned less bits but the amount isn't always higher for one of them. All methods show a certain trend where for example ONU 4 and 9 stand out. This trend follows the reverse distances shown in Figure 5.7 between the OLT and ONUs, here ONU 4 and 9 have the smallest distance to the OLT.

5.4.1.1.2 Case 2

In the next case we use the same distributed distance but change the traffic loads, those will be alternating between 150 and 50 Mbps. This change is made to better examine the stable matching method. In principle it should now favor the higher loaded ONUs over the lower loaded ONUs. Again the simulation is executed for all 4 assignment methods and the resulting chart can be seen in Figure 5.8.

The traffic loads for the ONUs is shown in Figure 5.9, as mentioned before it alternates between 150 and 50 Mbps.

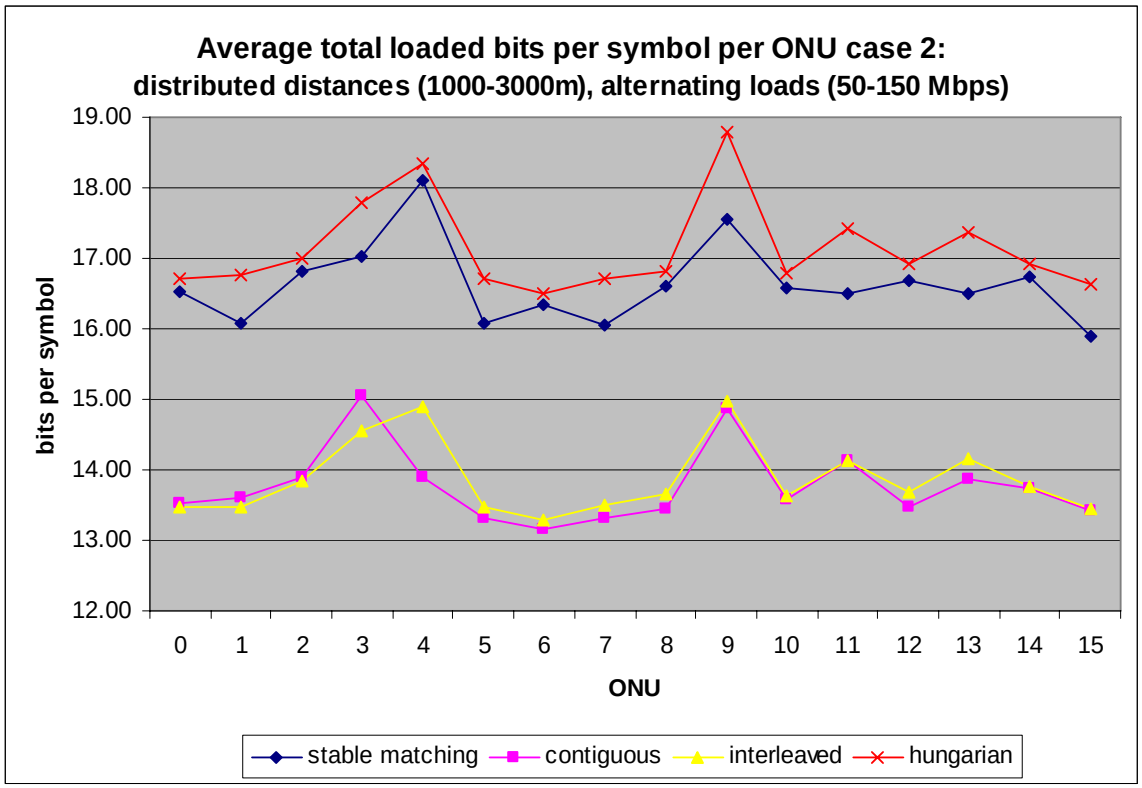

Figure 5.8

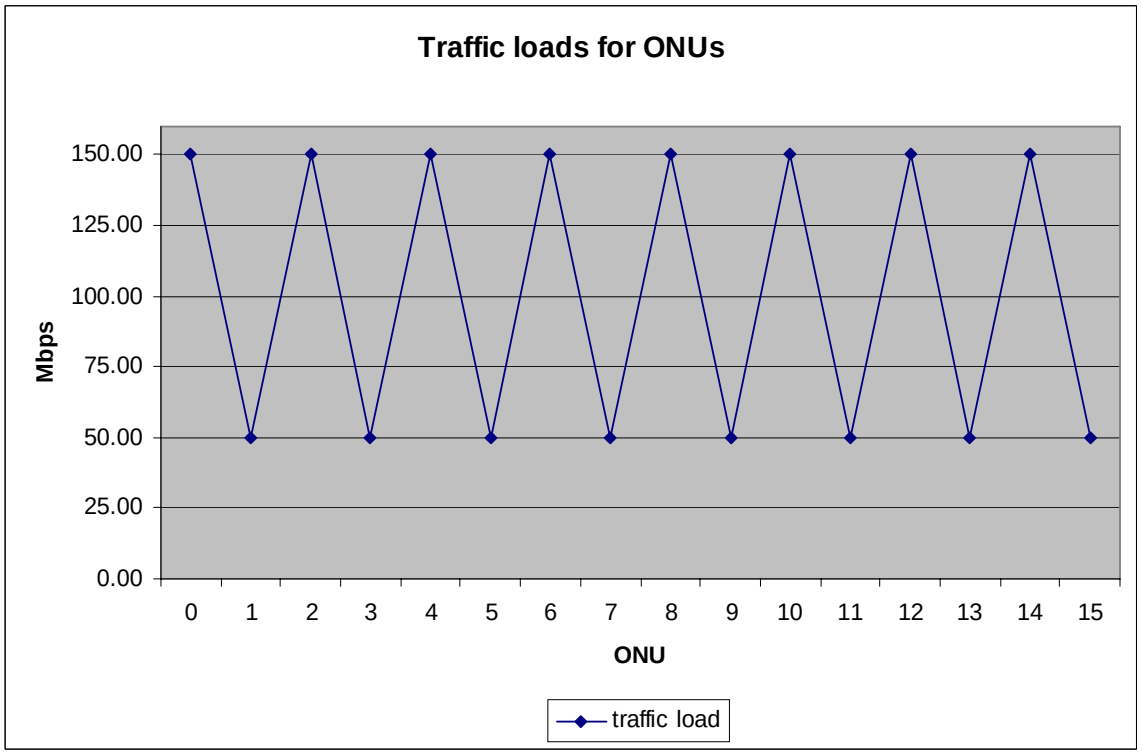

Figure 5.9

Now the even numbered ONUs have a higher traffic load then the odd numbered ONUs. The values for contiguous, interleaved and Hungarian loaded bits are exactly the same because they aren't affected by the traffic load. The values for the stable matching are different though, the even numbered higher loaded ONUs have a higher amount of loaded bits almost as high as the values for the Hungarian method. For the odd numbered, lower loaded ONUs the opposite is the case. The difference in the total amount of loaded bits over all ONUs is very small between the two cases, just the way they are spread out over the ONUs is rather different.

5.4.1.1.3 Case 3 / Case 4

To make the results clearer we rerun the two cases but now with fixed distances of 2000 meter between the OLT and the ONUs. In this way the trend of the chart isn't dependant on the difference in distances anymore. The results of this are shown in Figure 5.10 and Figure 5.11.

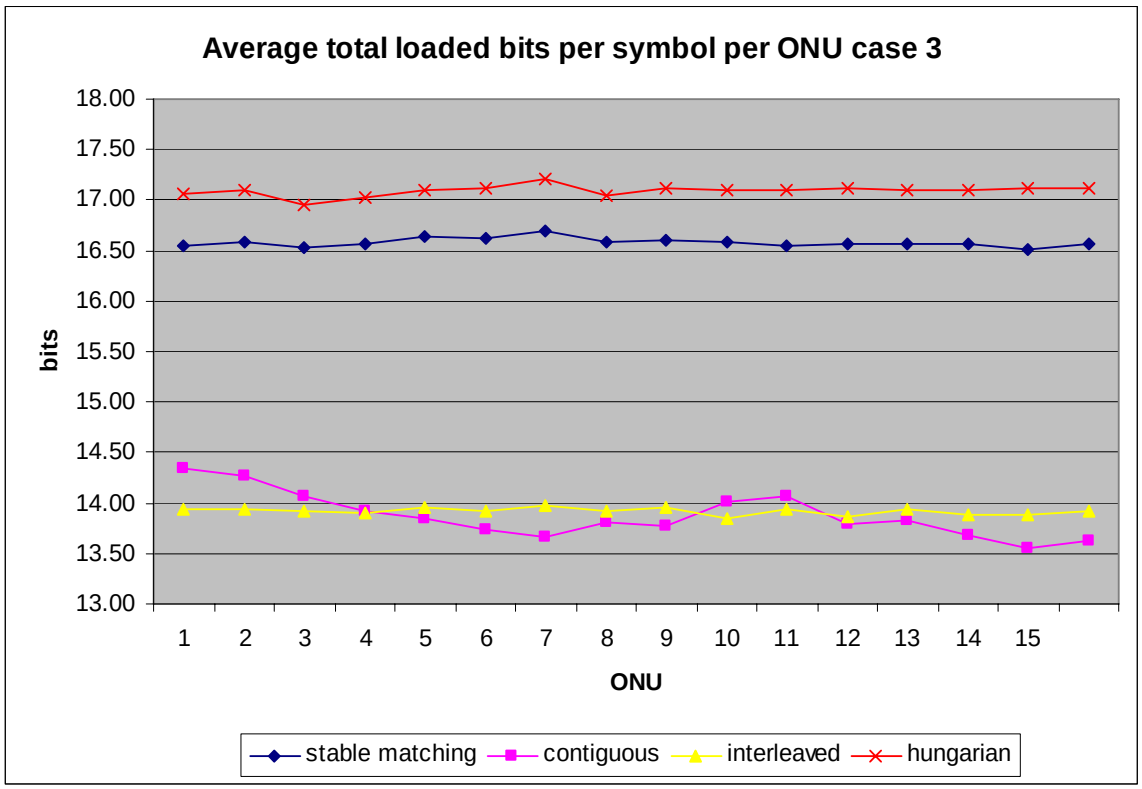

Figure 5.10

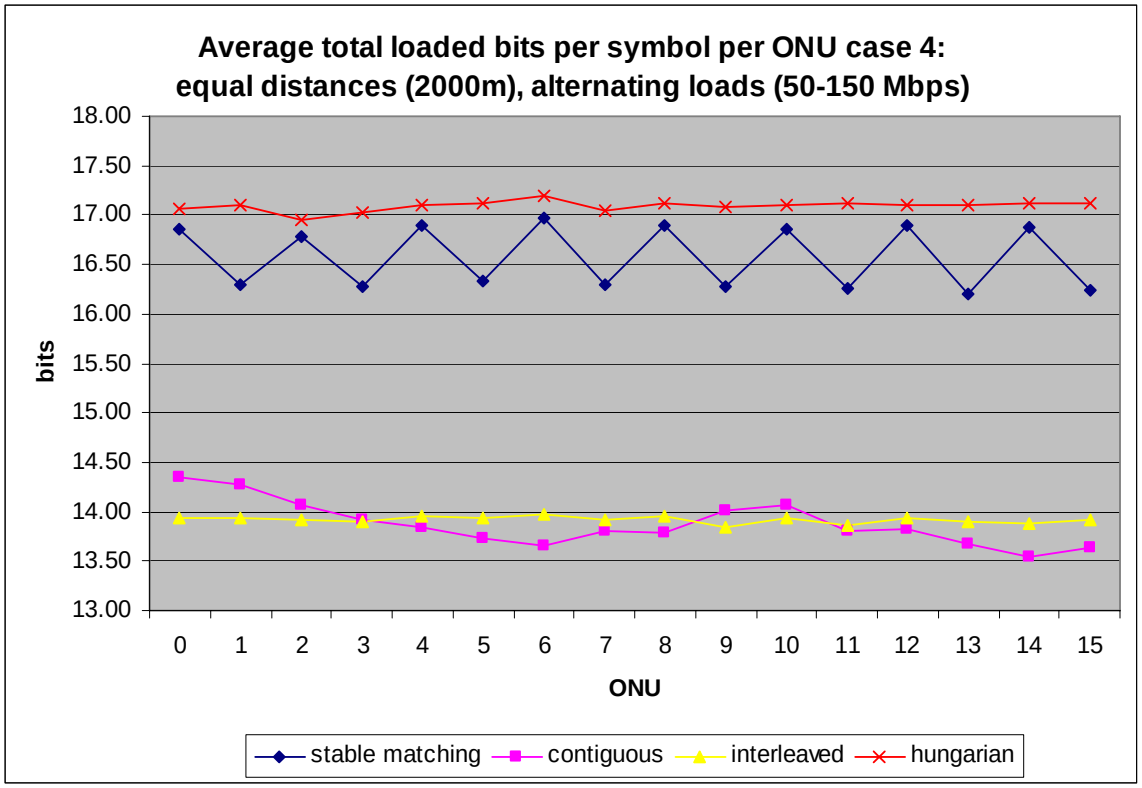

Figure 5.11

In the case with equal loads the order of which method can load the most bits doesn't change. Hungarian loads the most, closely follow by stable matching. Contiguous and interleaved follow some bits behind and the difference between them is very small and not better for one of them. The charts show less difference per ONU now because of the equal distances. The trend we see for all methods, though less profound, is caused by the difference in frequency responses.

A similar result is shown for the case with alternating loads and equal distances. Stable matching is able to come close to Hungarian for the higher loaded ONU. Here it is very clear to see that the loaded bits per symbol follow the traffic loads, which were shown in Figure 5.9.

5.4.1.2 Statistics

Next to just presenting the average values themselves we can perform some more statistics on them. The first statistic we want to perform is summing the average loaded bits per symbol per ONU for one assignment method. This can be seen as the total throughput for the system for a certain assignment method.

We now present those summed values for the 4 assignment methods in the 4 cases in Table 5.1.

Case	Stable matching	Contiguous	Interleaved	Hungarian
1	265.9159	220.2688	221.8803	274.2175
2	266.0993	220.2688	221.8803	274.2175
3	265.2683	221.997	222.6929	273.4851
4	265.2408	221.997	222.6929	273.4851

Table 5.1: Summed average total loaded bits per symbol

We see that that in all cases the summed value remains pretty much the same in all cases for all methods. For contiguous, interleaved and Hungarian there is no difference between case 1 and 2 and case 3 and 4. This is because these methods don't take the traffic loads into account and that is the only difference between the cases.

Let's look at the ratios between the different methods in relation to the Hungarian one, they are shown in Table 5.2. It's clear to see that in all cases the stable matching method is very close to the Hungarian one, at 97.0%.

Case	Stable matching	Contiguous	Interleaved	Hungarian
1	0.969726	0.803263	0.80914	1.0000
2	0.970395	0.803263	0.80914	1.0000
3	0.969955	0.811733	0.814278	1.0000
4	0.969855	0.811733	0.814278	1.0000

Table 5.2: Ratio's of different methods

5.4.1.2.1 Confidence intervals

As mentioned in section 5.3.5.2 confidence intervals can be calculated for a number of independent samples. This gives the interval in which for certain certainty the results lie. In the case of average loaded bits per symbol this is only useful for stable matching as for the other methods the result is the same for the different seeds. The result was that on average with 95% certainty the value is between 0.012 bits below and above the previously shown average, a pretty small interval. For the all the confidence intervals see appendix B.

5.4.1.3 Elasticity of stable matching

In the charts of the cases with alternating traffic loads we see that the stable matching method is able to adapt in some degree to the difference in traffic loads. The higher loaded ONUs get more loaded bits per symbol then the lower loaded ONUs, but as mentioned the summed value remains about the same. So this doesn't cause extra throughput to become available or to loose throughput. For now we focus on the case with equal distances because there it is easier to observe this effect of stable matching.

But how much difference is there in load bits per symbol between the higher and lower loaded ONUs? We take another look at case 4, as it shows the elastic effect of stable matching the best. When we average the values for both sets of loaded bits we get 16.88 for the higher loaded ONUs and 16.27 for the lower loaded bits, so the interval is 0.60 bits. This value isn't that large, compared to the 16.58 loaded bits per symbol on average over all ONUs. But it does enable stable matching to come close to the Hungarian method in certain circumstances. And this elasticity effect can be useful in handling irregularly occurring large bursts of traffic.

A logical question now is; can we enlarge this interval of the stable matching assignment method? Under what circumstance might the system offer a larger interval? Let's examine the cases with less higher loaded ONUs. For this, we use case 5 and case 6, with respectively 4 and 1 higher loaded ONU(s). Because interleaved and contiguous are independent of the traffic load and always perform worse then stable matching we leave those out here. Hungarian is added though as a reference, but as it's also independent of the traffic load it will be the same as in cases 3 and 4.

For case 5 and 6 the resulting charts are respectively shown in Figure 5.12 and Figure 5.13.

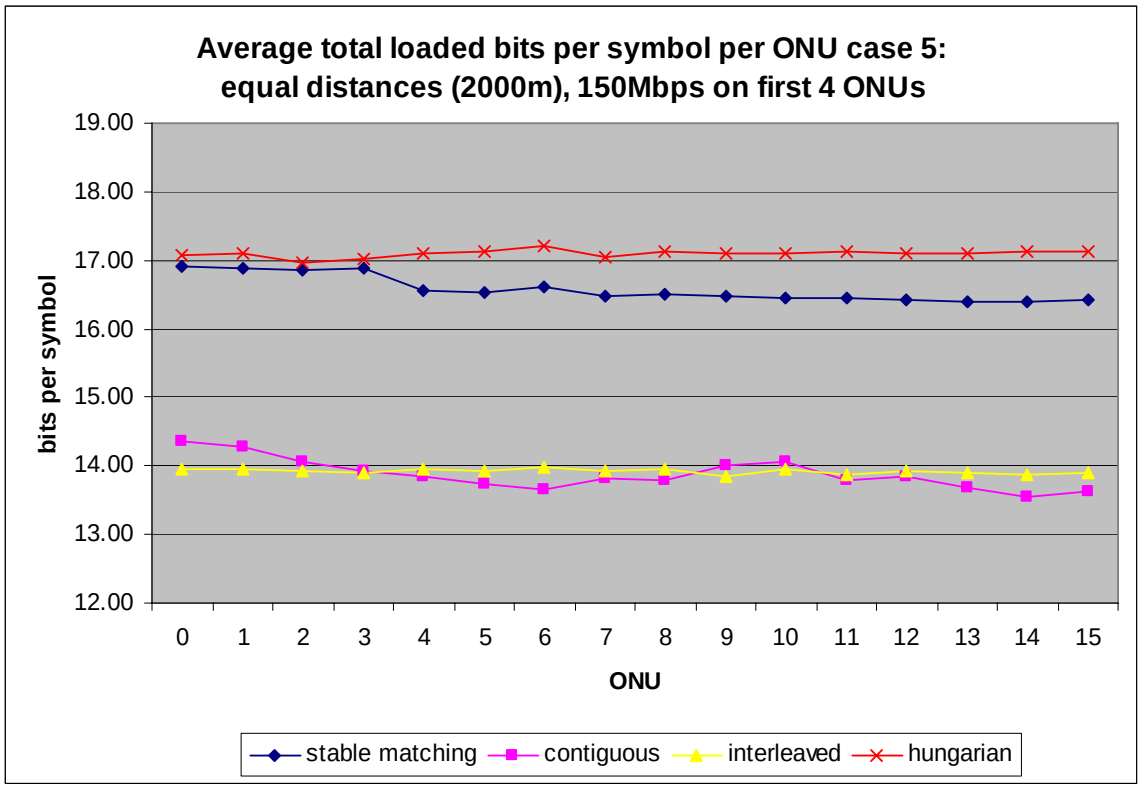

Figure 5.12

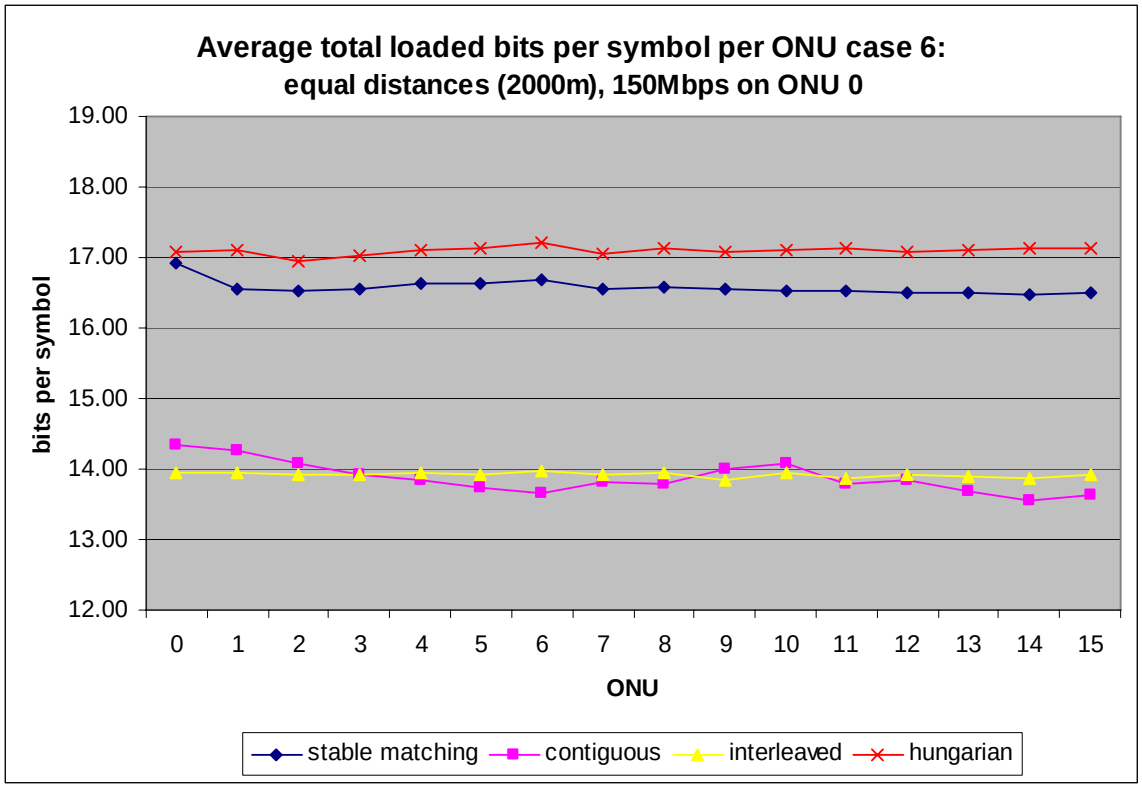

Figure 5.13

The related statistics are given in Table 5.3 and Table 5.4, we also repeat the values for case 4.

Case	Stable matching	Hungarian
4	265.24	273.49
5	265.26	273.49
6	265.21	273.49

Table 5.3: Summed average loaded bit per symbol

Case	Avg loaded bits high loaded ONU(s)	Avg loaded bits low loaded ONUs	Interval	Avg loaded bits all ONUs
4	16.88	16.27	0.61	16.58
5	16.88	16.48	0.41	16.58
6	16.93	16.55	0.38	16.58

Table 5.4: The effect of less high loaded ONUs

Overall it doesn't appear to matter much how many ONUs are highly loaded. The maximum average amount of loaded bits per symbol seems to be around 16.90. Another possible way to enlarge the interval is by increasing the traffic loads of the higher loaded ONU(s). This has as effect that the higher loaded ONUs will have a larger queue size all the time and therefore are always put in front of the preference lists of the subcarriers. When an ONU is less highly loaded its queue length can become the same or

smaller as the lower loaded ONUs. Then such an ONU might not be recognized as a higher loaded ONU and therefore doesn't get a better set of subcarriers.

On itself this is correct behavior, but if we want to examine how to increase the interval we don't want it to happen. So some variants of case 6, but with a higher traffic load for the higher loaded ONU, were explored and it was indeed visible that the interval got a little larger. The most extreme of these variants, case 8 was with 250 Mbps on one ONU and 50 Mbps on the others; this resulted in 17.00 loaded bits per symbol for that user.

In both approaches the interval can be stretched a little more, but it's not that spectacular. Isn't it simply possible to get more then around 17.00 bits for a certain user? This leads to the question, what is really the maximum for a specific ONU, without being unfair to the other ONUs? To answer that question a separate assignment method was developed, called the unfair method. In this method one ONU, the selected ONU, can select the best 8 possible subcarriers, from its point of view, for itself. The other subcarriers are then given to the other ONUs in no specific order. This means that the selected ONU gets the best subcarriers and therefore should be able to get the highest amount of total loaded bits per symbol during every time epoch for that specific ONU. The simulation was executed using this subcarrier assignment method and that resulted in an amount of bits per symbol below that of case 8 for the higher loaded ONU, about 16.30 bits per symbol.

Why did this happen? To understand this we need to look at the bitloading process. As mentioned these were assigned in no specific order, so there is a chance bad subcarriers are assigned. As we are simulating downlink traffic the available power in the bitloading stage is used for all subcarriers, so besides the 8 subcarriers assigned to the selected ONU, also on the 120 subchannels which are just assigned to ONUs without regard of their quality for that ONU. During the bitloading the 8 subcarriers assigned to the selected ONU get their 2 bits easily. The rest of the power will be used on loading the first 2 bits on subcarriers that can require more power then when these subcarriers would have been assigned based on quality like with stable matching. While these subcarrier require more power to load 2 bits, it is not so much power that it would be cheaper to load 2 more bits per symbol on the 8 subcarriers assigned to the selected ONU. So in the end the randomly assigned subcarriers require too much power for the 8 subcarriers assigned to the selected ONU to get loaded to their possible maximum.

So when looking at the bits per symbol we determine the unfair method doesn't perform better. Therefore we can't draw any useful conclusions about the amount of bits per symbol the stable matching method is able to load. But when looking at the actual assigned subcarriers we see something interesting, these are almost identical between the stable matching method in case 8 and the unfair method.

We say almost because there are two exceptions. The first one is during the first time epoch, which assignment is determined before traffic starts so in stable matching the heavy loaded ONU isn't in front of the preference list yet. The other exception is when there are more then one subcarriers whose associated subchannels have the same normalized SNRs. When these subcarriers are around the 8^{th} best it is possible that there

is a difference between the one assigned by the unfair algorithm and the one assigned by the stable matching. This is caused by the fact that with the unfair algorithm the subcarriers are sorted and a lower numbered subcarrier will be in front of higher numbered subcarrier whose associated subchannels have an equal normalized SNR. With stable matching the ties of these subchannels with equal normalized SNR have been randomly broken. So while different subcarriers are assigned the normalized SNRs of the associated subchannels are equal.

We can conclude that when an ONU has the longest queue length, and therefore is in front of the preference lists of the subcarriers, it behaves as the unfair method in respect to the normalized SNR of the associated subchannels of its 8 subcarriers.

This also shows that the subcarrier assignment and bitloading process is a complicated one in which several things depend on each other. We can limit the complexity somewhat in the uplink direction. Among other things, then every ONU has its own amount of available power for bitloading, so there is no effect caused by the subcarriers assigned to other ONUs. If we run the simulation for traffic in the uplink direction we see that the total bits per symbol loaded with unfair method surpasses that of the Hungarian method for the chosen ONU. The stable matching method results in the same amount of loaded bits per symbol for the most highly loaded ONU during the whole simulation. This is what we expected as the Hungarian method tries to find an overall optimal solution without actively favoring a certain ONU.

5.4.1.4 Equalization effect

Another measure of interest is the variance between the loaded bits of the different ONUs. This is shown in Table 5.5 and can be seen as expressing with a number how flat a certain line in the previously shown Figure 5.6, Figure 5.8, Figure 5.10, Figure 5.11.

Case	Stable matching	Contiguous	Interleaved	Hungarian
1	0.309924	0.285667	0.277138	0.42628
2	0.320073	0.285667	0.277138	0.42628
3	0.001960	0.051109	0.001333	0.00283
4	0.100077	0.051109	0.001333	0.00283

Table 5.5: Variance in the average total loaded bits per symbol

This table doesn't show anything particularly interesting. For the distributed distances cases the variance is larger compared to the equal distances cases. This is because the difference in distance has a clear effect on the amount of bits per symbol that can be loaded. In case 4 the variance for stable matching is larger then in case 3 because of the adaptation to the traffic load effect.

But what if we increase the traffic load on all ONUs? This will have as an effect that the stable matching algorithm can better notice the difference in distances. The ONUs that are further away from the OLT will initially be loaded with less bits per symbol. That is

because their normalized SNR is worse then that of the ONUs that are closer to the OLT. As they have less loaded bits per symbol their queue lengths will grow. Then the higher queue length causes these ONUs to be more in the front of the preferences lists. Therefore these ONUs will be selected more easily in the stable matching method and get more loaded bits per symbol, thereby (partly) negating the effect of their worse normalized SNR.

In case 10, 11 and 12 all ONUs are loaded with respectively 150 Mbps, 175 Mbps and 200 Mbps on all ONUs, case 1 with 100 Mbps on all ONUs is added as a reference. The resulting variances for stable matching and Hungarian are shown in Table 5.6.

Case	Mbps	Stable Matching	Hungarian
1	100	0.30992	0.42628
10	150	0.27531	0.42093
11	175	0.00831	0.42093
12	200	0.00778	0.42093

Table 5.6: The equalization effect

To make things more clear we show the chart on which these variances are based in Figure 5.14. Hungarian is just shown once as it doesn't change with the different traffic loads. We added some extra lines to show the effect in the range up to the point that the method starts getting overloaded.

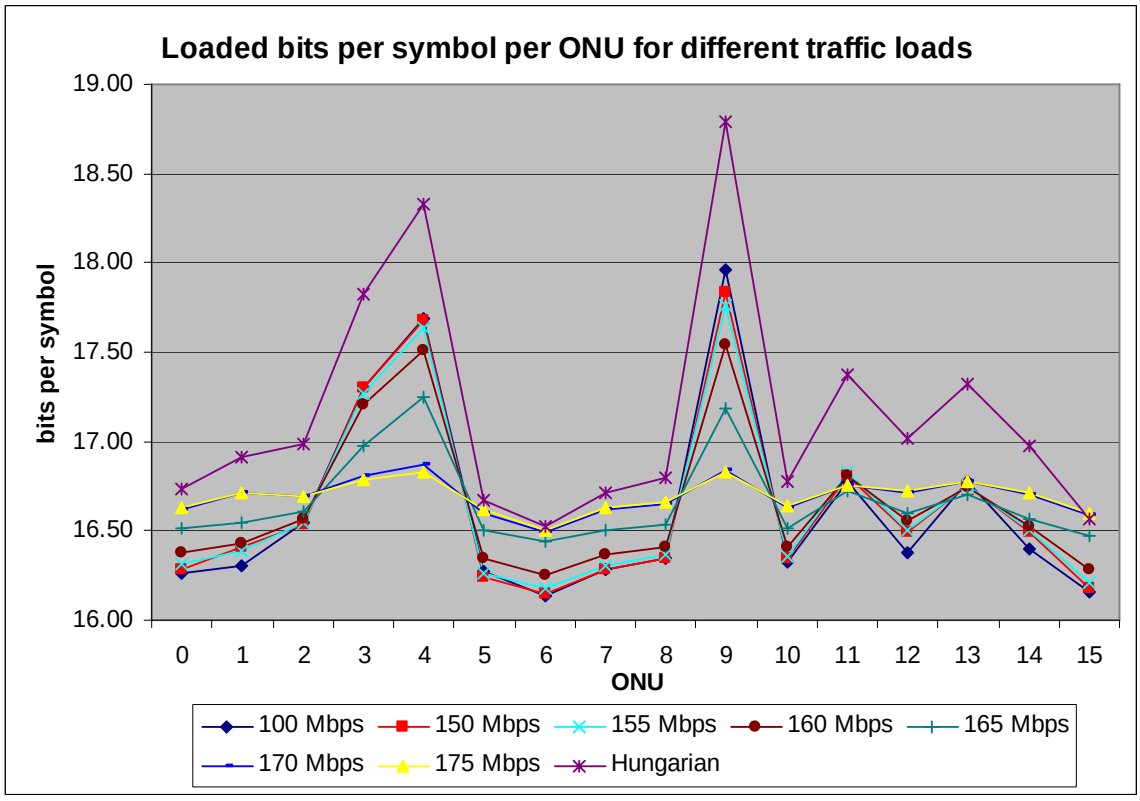

Figure 5.14

In Table 5.6 the variance for the Hungarian changes after case 1 because we only performed simulations over 576 time epoch for the other three cases. This was done as these simulations are very demanding on the computer they run on and therefore take a long time to finish.

From Table 5.6 and Figure 5.14 we can see that for a higher load on all ONUs the variance for stable matching decreases. The higher points and lower points in the chart move towards each other, up to almost a flat line. This means that the difference in loaded bits per symbol per ONU becomes smaller. So even though there is a difference in distances and therefore normalized SNRs the stable matching method gives the ONUs a more equal amount of loaded bits per symbol for higher loads.

5.4.2 Sojourn times

The amount of average total assigned bits gives us a certain insight in the assignment algorithms, but it doesn't show the whole picture (more on this further in this section). To be able to compare the algorithms further we look at the packet sojourn time.

5.4.2.1 Mean sojourn times

One such sojourn time on itself isn't that useful, but the average value of all sojourn time per ONU samples for an execution of the simulation is, we call that the mean sojourn time. As with the loaded bits per symbol here we also execute the simulation for 5 data sets and 15 OPNET seeds per dataset and average over the 75 results.

While using the loaded bits is pretty straight forward, working with the sojourn times is somewhat more complicated. The first big difference is the number of samples, there are only 8 loaded bits values per ONU per time epoch. But there are many more sojourn time values during one time epoch per ONU. It does depend on the traffic load, but even with a low traffic load there will be much more samples.

Because the mean sojourn times have an initial transient period (mentioned before in section 5.3.5.1 initial transient removal is used in the results of the sojourn time. The sojourn time samples from the first 5 seconds are ignored.

5.4.2.1.1 Case 1

Let's first look at the mean sojourn times for the case 1, it is shown in Figure 5.15. Like with the loaded bits, the dots show the actual data points and the connecting lines are used to better see the behavior. In these charts time is the scale on the x-axis in seconds and it represents sojourn time, and usually the lower is the better. So the lowest line in the chart represents the most desirable outcome.

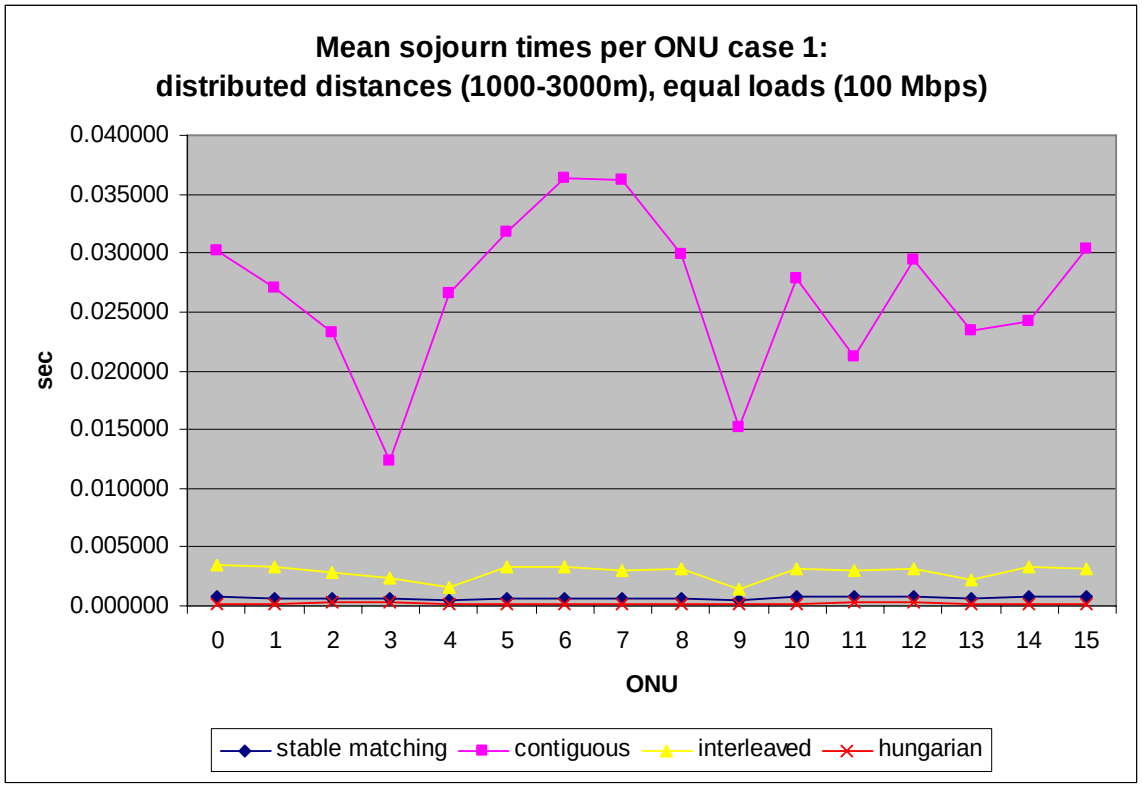

Figure 5.15

This chart shows two things, first that the contiguous method has pretty bad mean sojourn times. While we saw little to no difference in the amount of loaded bits between contiguous and interleaved, this chart shows a whole different situation. The reason for this is that with the contiguous method it can more easily happen that all subcarriers assigned to a certain ONU are so bad so that no bits can be loaded for some time epochs. This is because the link characteristic is like shown in Figure 5.2, and with the contiguous method it is possible to get in such a dip with all assigned subcarriers. The instances where for a whole time epoch no data can be send are very bad for the mean sojourn time and causes the effect shown in the figure.

The second observation is a consequence of the first; the mean sojourn times for the contiguous method are much larger in relation to the other methods, therefore making it hard to see the other values. So we won't show those values in the charts anymore, but as they are usually much worse then the other methods they aren't that useful. We will keep track of them and when something interesting shows up it will be shown. The results without the contiguous method are shown in Figure 5.16.

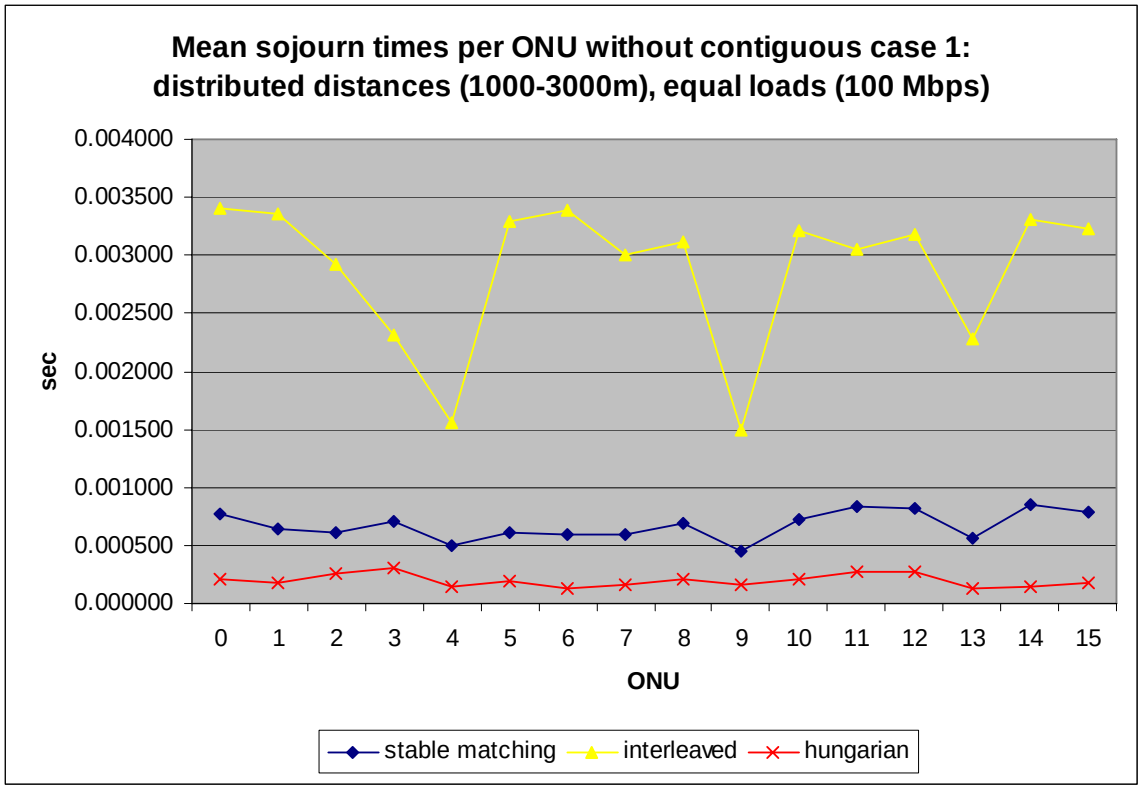

Figure 5.16

5.4.2.1.2 Case 2

In Figure 5.17 the chart is shown for case 2, again contiguous causes the chart to be rather unreadable. Though here interleaved also produces too different results and is removed from Figure 5.18. The associated traffic loads for the ONUs can be found in Figure 5.9.

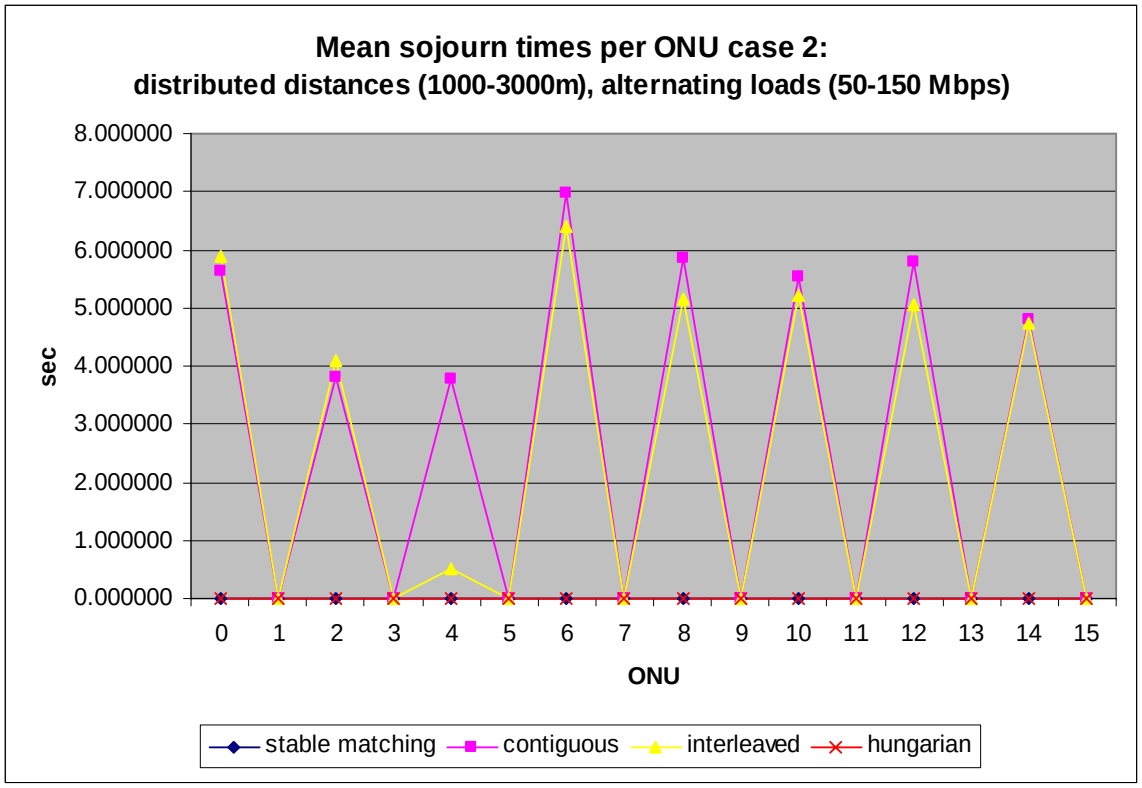

Figure 5.17

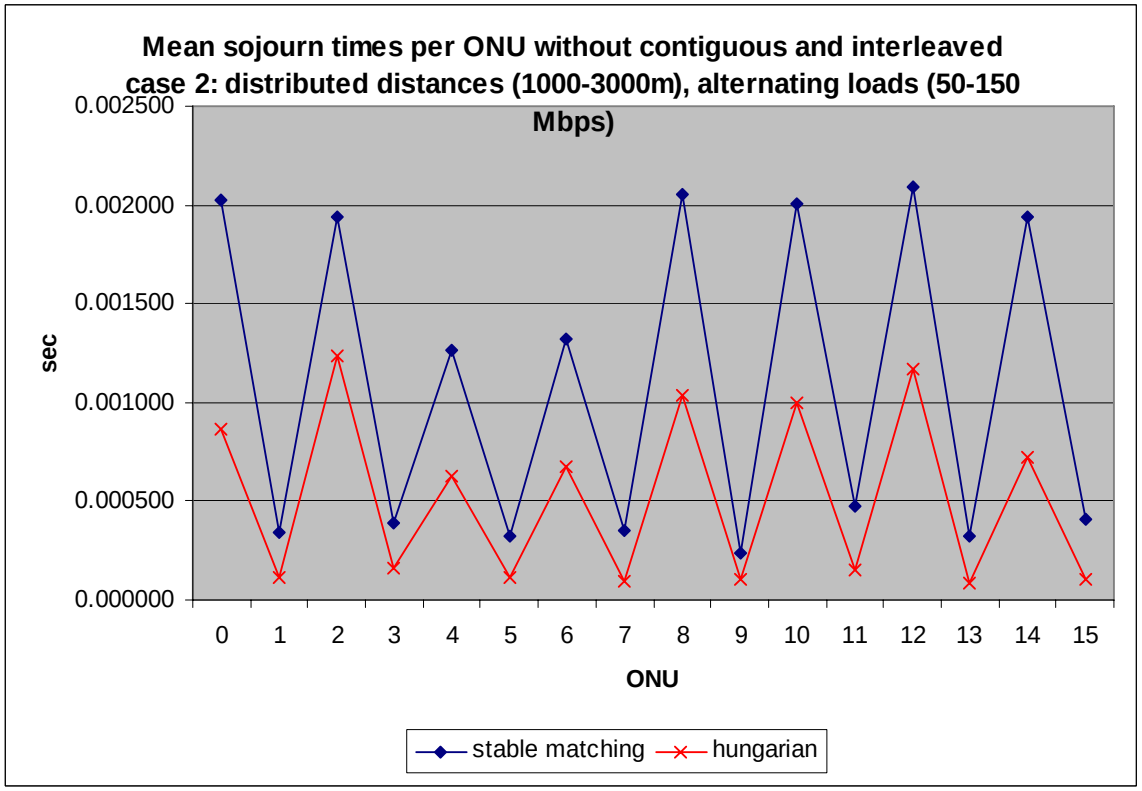

Figure 5.18

Both semi static methods clearly perform worse in the cases with the higher loaded ONUs. This is because 150 Mbps is close to what these methods can handle based on the amount of total bits per symbol that are loaded. So there will be time epochs when they don't have enough capacity and the mean sojourn time goes up because of that.

For stable matching and Hungarian the higher loaded ONUs also show a higher mean sojourn time. This is a direct result from the traffic load, more traffic mean larger queue sizes.

5.4.2.1.3 Case 3 / case 4

We show the mean sojourn times for case 3 and 4 in Figure 5.19 and Figure 5.20. Both are without the contiguous and interleaved values as these make the charts less readable and aren't that interesting anyway.

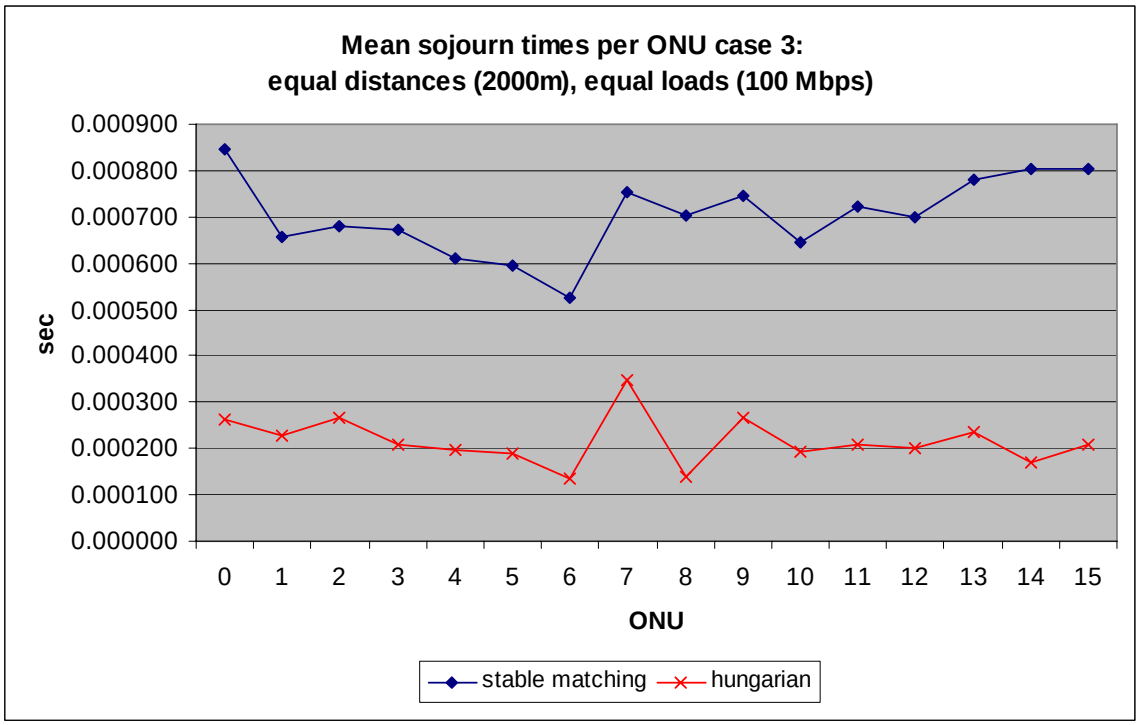

Figure 5.19

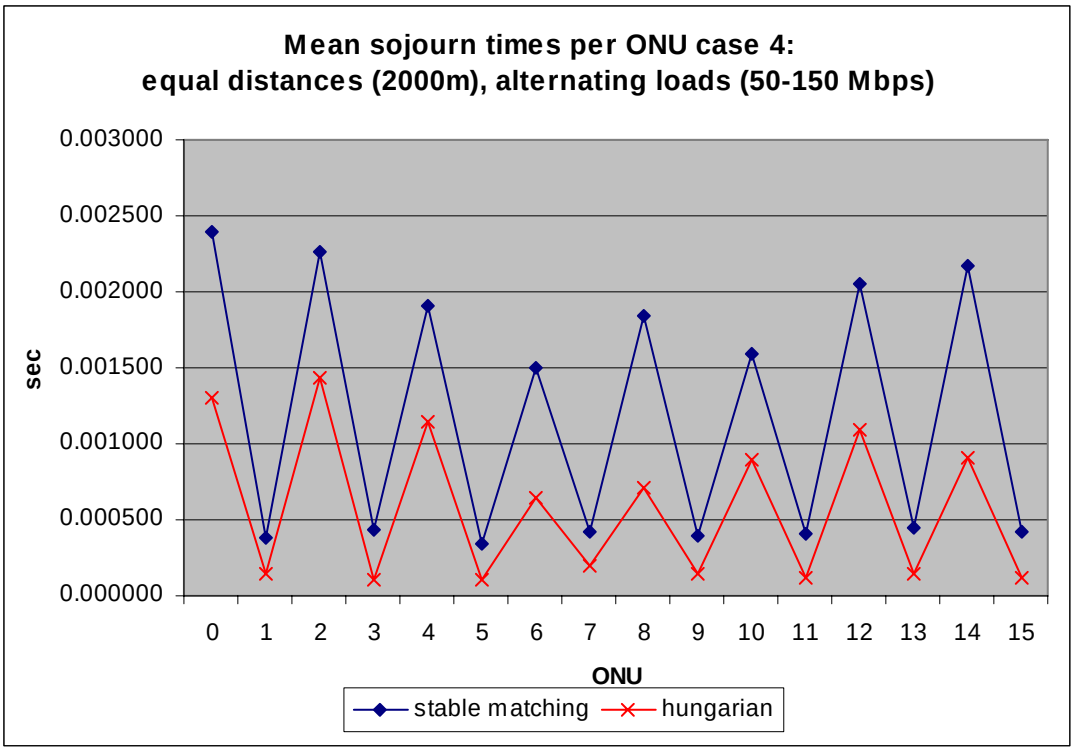

Figure 5.20

The difference in values for case 3 seems to make sense; Hungarian is able to load more bits per symbol so it's getting better sojourn times. But in case 4 one would expect stable matching to be closer to the sojourn times of Hungarianin the cases with higher loaded bits.

Is this difference explainable? For this we have to look back at the loaded bits per symbol. But now the distribution of the total loaded bits per symbol over all the time epochs are examined. This means how many times a certain amount of total bits per symbol is loaded during a time epoch for one simulation run. Examples can be seen in Table 5.7and Table 5.8, they show the results from one simulation run of the stable matching method and Hungarian method for one ONU.

bits	times	bits	times	bits	times	bits	times
0	3	18	11	34	0	50	0
2	1	20	15	36	0	52	0
4	0	22	6	38	0	54	0
6	0	24	11	40	0	56	0
8	0	26	11	42	0	58	0
10	1	28	4	44	0	60	0
12	1	30	8	46	0	62	0
14	3	32	27	48	0	64	0
16	973						

Table 5.7: Number of times a total number of bits per symbol is loaded for stable matching

bits	times	bits	times	bits	times	bits	times
0	0	18	12	34	0	50	0
2	0	20	13	36	0	52	0
4	0	22	9	38	0	54	0
6	1	24	8	40	0	56	0
8	0	26	4	42	0	58	0
10	0	28	6	44	0	60	0
12	1	30	11	46	0	62	0
14	0	32	40	48	0	64	0
16	970						

Table 5.8: Number of times a total number of bits per symbol is loaded for Hungarian

As can be seen the maximum amount of bits per symbol for one ONU is 32. This means that all 8 subcarriers were loaded with 4 bits per symbol. The bit loading algorithm is able to load up to 8 bits on one subcarrier (so 64 bits per symbol for one ONU) but with the characteristics (i.e. optical power, frequency response) of this system, 4 bit per symbol per subcarrier is the limit.

For the stable matching method it is clear that the in over 75% of the time epochs 16 bits per symbol are available for an ONU. After that 32 bit per symbol appears the most, followed by occurrences from the range between 18 and 30 bits per symbol. Situations in which less then 16 bits are available occur the least, in such a situation the bit loading algorithm was unable to load any bits on at least one subcarrier.

How does this explain the difference in sojourn times and loaded bits per symbol? When all the number of times are multiplied with the related amount of loaded bits per symbol (i.e. 1 * 2, 1 * 4, 0 * 6, 2 * 8, ...) we get a higher total for stable matching then when we do that for the Hungarian based method. This corresponds with what we see in the chart, shown in Figure 5.11 of the loaded bits per symbol for case 4. But why do the mean sojourn times for the same case show a different effect in Figure 5.20?

This is caused by the difference in how often during a simulation run the number of available bits per symbol for one ONU in a time epoch is below 16. For the stable matching method this occurs more often then for the Hungarian based method, as can be seen when comparing both tables. These situations also have a stronger effect on the mean sojourn time compared to when there are more bits available.

Aren't there situations when the stable matching assignment method is able to get better mean sojourn times? It is able to come close to the amount of loaded bits per symbol of Hungarian, so is it able to get better mean sojourn times? After some experiments we were unable to find such occurrences though it comes close.

When we set the load for one ONU to 200 Mbps as in case 7 (and 50Mbps on the other ONUs), we get mean sojourn times as shown in Table 5.9 for the different methods. The values for case 6 are added as a reference, it can be seen that for 150 Mbps on one ONU Hungarian has a better mean sojourn time than stable matching, though the difference is small. Contiguous and interleaved are already overloading in case 6, they can't handle the 150 Mbps traffic load.

Case	Mbps on ONU	Stable matching	Contiguous	Interleaved	Hungarian
6	150	0.00184	2.57549	3.97617	0.00131
7	200	8.32093	15.8932	16.9673	8.12931
8	250	17.8145	23.9137	24.7690	17.6762

Table 5.9: Mean sojourn times for a higher loaded ONU

With 200 Mbps on one ONU, stable matching and Hungarian are both overloading, but because of the less bits per symbol for stable matching it overloads a little earlier. Contiguous and interleaved only get overloaded more then before.

We can raise the traffic load for the one ONU even more, for example to 250 Mbps in case 8. Here for all methods the effect is only worse, they are even more overloaded. The difference between stable matching and Hungarian is still small, but Hungarian remains the best.

6 Conclusion and future work

6.1 Discussion of the results

Let's first recall the main goal of this assignment: "Develop and simulate an adaptive subcarrier assignment algorithm to be used in a full service access network using multimode fibre".

After an exploration of the subcarrier assignment problem, we found the stable matching algorithm. The properties it has - always resulting in a stable matching, getting the best possible "partners" for the selecting entities and being able to take the preferences of both sets of entities into account - appeared to be useful for a subcarrier assignment method. There was a problem of unequal sets to solve, but several solutions for that were presented.

Before the actual behaviour of the stable matching subcarrier assignment method can be determined, a simulation had to be developed. For this OPNET was used. In it a simple network with associated fibre frequency response was modelled. Then the stable matching and three other subcarrier assignment methods (contiguous, interleaved and Hungarian) were implemented for simulation.

Once this was all done, the behaviour of the stable matching subcarrier assignment method along with the other assignment method can be simulated and a performance comparison could be done. From this process we can draw the following conclusions about the stable matching subcarrier assignment method.

- The method always creates subcarrier assignments that fulfil the defined demand (assigning 128 subcarriers evenly over 16 ONUs and assigning each subcarrier to only one ONU).
- The method is able to select the best possible subcarriers for a single ONU, in its point of view, when that is the most heavily loaded ONU (the ONU with the largest queue length, so in front of the preference lists of the subcarriers).
- The method shows an elasticity effect: when confronted with different traffic loads it will load more bits per symbol on the higher loaded ONUs.
- The method shows an equalizing effect: when the traffic load is perceivably high for some ONUs, the difference in loaded bit per symbol per ONU caused by the variation in distances between ONUs and OLT is smaller than when those ONUs are lowly loaded.

From this we can conclude that we indeed have developed an adaptive subcarrier assignment algorithm to be used in a full service access network using multimode fibre. A paper [TaSi06] on this method with preliminary results was published during NOC 2006.

6.2 Future work

A thesis can never be exhaustive; there are always areas that could have used some more attention and new questions that pop up during the thesis. Unfortunately time is limited so it is not possible to do all that research and answer all the questions. Therefore we now list what we feel are the things that could be explored further.

There is a distinct difference to make in the different directions this future work can go. The first direction is that of the larger project and eventually the real life system. Everything done in this thesis was with the idea that it could become part of the larger project. So the best future work is actually putting this in the larger system once that is finished.

But as that probably isn't going to happen in the nearby future, there is another direction to go: enhancing the simulation. A simulation is always an abstraction of a real system and usually can't match it exactly. In this case it was also the first time that the different aspects of this system were explored. But now that there is a base on which to build, it's possible to enhance certain aspects. Future work can be, for example, creating more realistic traffic generation.

Also within the chosen subcarrier assignment method future work is possible. As can be seen in chapter 3, there are many types of subcarrier assignment methods. For some it's already clear they don't meet the demands we have set for possible assignment methods, but others might be possible, either with or without extra changes.

Even within the stable matching approach there are other possibilities to explore. For the stable matching algorithm the "virtual user" variant was used. But as mentioned in chapter 4 another approach is the "free user" one, which might be able to give some more elasticity. On the area of elasticity future work would indeed be welcome, as this would certainly make the method more effective. One possibility to examine here is allowing more flexibility in the amount of subcarrier assigned per ONU. Now that was fixed to 8 per ONU, but with determining this dynamically based on the traffic demands a solution much closer to the optimal one is possible.

References

[Bria04] A C implementation of the Hungarian Method, B. P. Gerkey, http://ai.stanford.edu/~gerkey/tools/hungarian.html (last checked 25-06-2006).

[Cyri04] C-implementation of the Hungarian Method, C. Stachniss, http://www.informatik.uni-freiburg.de/~stachnis/misc.html (last checked 25-06-2006).

[EtPl00] W. van Etten and J. van der Plaats, Principles of optical fibre communication, 2000, Twente university press, Enschede.

[GaSh62] D. Gale and L.S. Shapley, College admissions and the stability of marriage, American Mathematical Monthly, 69:9-15, 1962

[Google] Google, http://www.google.com/ (last checked 31-07-2006).

[GuIr89] D. Gusfield and R.W. Irving, The Stable Marriage Problem: Structures and Algorithms, The MIT Press, 1989

[IOP-GenCom]
Full-service Access Network using Multimode Fibre,
http://www.ctit.utwente.nl/research/projects/national/iop-gencom/fsan.doc/ (last checked 24-06-2006).

[KiLe01] I. Kim, H. Leem Lee, B. Kim, Y. H. Lee, On the Use of Linear Programming for Dynamic Subchannel and Bit Allocation in Multiuser OFDM, 2001, IEEE.

[Knut93] D. E. Knuth, The Stanford GraphBase: A Platform for Combinatorial Computing, 1993, ACM Press, New York.

[KoBo03] T. Koonen, H. van den Boom, G.D. Khoe, Broadband Access and In-House Networks - Extending the Capabilities of Multimode Fibre Networks. Proc. of ECOC, 2003.

[Kuhn55] Kuhn, H. W. (1955), "The Hungarian method for the assignment problem", Naval Research Logistics Quarterly, 2:83–87.

[Munk57] Munkres, J. Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics vol 5 no 1 p 32 (Mar. 1957), 32-38

[MuPf02] G. Munz, S. Pfletschinger, and J. Speidel, "An efficient waterfilling algorithm for multiple access OFDM," Proc. IEEE Global Telecommunications Conf. (GLOBECOM 2002), vol.1, pp.681–685, Nov. 2002.

[RaWh99] L. Raddatz, I.H. White, Overcoming the Modal Bandwidth Limitation of Multimode Fiber by Using Passband Modulation, Photonics Technology Letters, IEEE, 1999.

[RhCi01] W. Rhee, J. M. Cioffi, Increase in Capacity of Multiuser OFDM System Using Dynamic Subchannel Allocation, 2001, IEEE.

[Springer] SpringerLink, http://www.springerlink.com (last checked 31-07-2006).

[TaBo05] R.O. Taniman and A.C. van Bochove, Duplexing methods for PON systems using multimode fiber with multicarrier transmission, Proceedings Symposium IEEE/LEOS Benelux Chapter, 2005, Mons.

[TaSi06] R.O. Taniman, B. Sikkes, A.C. van Bochove, P.T. de Boer, Stable-matching-based subcarrier assignment method for multimode PON using a multicarrier variant of subcarrier multiplexing, 2006, NOC.

[Wils72] L.B. Wilson, An analysis of the stable marriage assignment algorithm, BIT, 12(1972), pp. 569-575.

[Wins04] W.L. Winston, Operations Research: Applications and Algorithms, 4th edition, 2004, PWS-KENT Publishing Company, Boston.

[YuCi01] W.Yu and J. M. Cioffi, “On constant-power water-filling,” in Proc. IEEE Int. Conf. Communications (ICC 2001), vol. 6, Helsinki, Finland, June 11–14, 2001, pp. 1665–1669.

Appendix A OPNET

In this appendix we discuss the simulator used in this thesis, OPNET version 11.5. We start with giving some general background of OPNET and how modeling / simulating in OPNET works. Then we discuss the actual implementation we made in OPNET.

1 Introduction

OPNET Modeler is a network modeling and simulation tool developed by OPNET Technologies, Inc. It is a so called discrete event simulator; this means that it keeps a list of events and times when these events are scheduled to happen. During execution it will process the event list and when the internal simulation time is equal to the scheduled time for an event the simulator will execute it. Events themselves can schedule new events to simulate a system with recurring events. Because the simulator jumps from event to event and thereby skipping the time in which nothing happens it is possible to simulate long time periods in a fraction of that time.

The OPNET Modeler software is being used by a wide variety of companies and research institutions. It contains a wide range of pre-defined network elements like routers, switches, workstations and links from different vendors. But it is also possible to create a new element from scratch. With these elements it is possible to create all kind of networks. The network scale can be ranged from worldwide till just a few meters.

Next to wired links it is possible to use wireless links and even satellite links. For the wireless links the terrain they function on can be modeled and for the satellite links their orbits can be modeled. Overall it's an extremely extensive modeling tool of which only a limited portion will be used for this thesis.

2 Modeling in OPNET

Modeling a system in OPNET is done in a three level hierarchical fashion, as shown in figure 1. At the top level is the network level, there different nodes and links can be placed via the project editor. This is a graphical editor in which via a drag and drop principle the required network can be created. A project can contain different scenarios with totally different network layouts or just different settings for certain network elements.

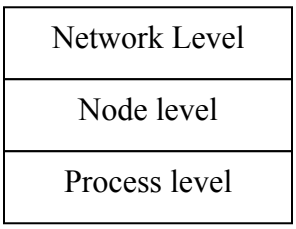

Figure A.1: The three OPNET levels

Below the network level there is the node level, consisting of different modules. The modules are also modeled via a graphical editor. Modules can be for example transmitters, receivers, queues or generic processors which functionality a user can completely define. Sending packets from module to module is done via packets streams. To read values from one module in another or to set values in one module from another statistic wires can be used.

At the bottom is the process level, consisting of a combination of finite state machines and c/c++ code. Via the process editor it is possible to define states and transitions between those states. Then programming code can be created that should be executed when a state is entered, exited or when a certain transition is made. This code is normal c or c++ and can be enhanced with a wide array of functions offered by the Modeler software. Those are function to for example send packets, take measurements on queues or wireless links, log statistics and so on.

3 Simulating in OPNET

Once a system had been modeled its behavior can be simulated. It is possible to run the same modeled system with different input settings so that effect of those can be examined. OPNET offers an interface in which it is possible to manage these settings and to queue them in a sequence. In this way it is possible to execute a number of simulation runs with one click instead of starting them all separately.

Results from a simulation can be obtained by having the simulation itself output them via the screen or files. OPNET itself offers some ways also, via statistics or probes. With statistics a value is written to a statistic handle during the simulation and these values can be recorded. With probes values can also be recorded but with some processing being performed, for example determining the average. When the results are recorded by OPNET they can be presented in graphs when the simulation has ended or stored to be examined at a later stage.

4 Modeling the system under study in OPNET

We will now describe the modeling in OPNET of the network used in this thesis. Starting at the project level; figure 2 shows the network model and scenario of the project. In the center there is one OLT and surrounding it are 16 ONUs. There are no links because no packets are actually transmitted.

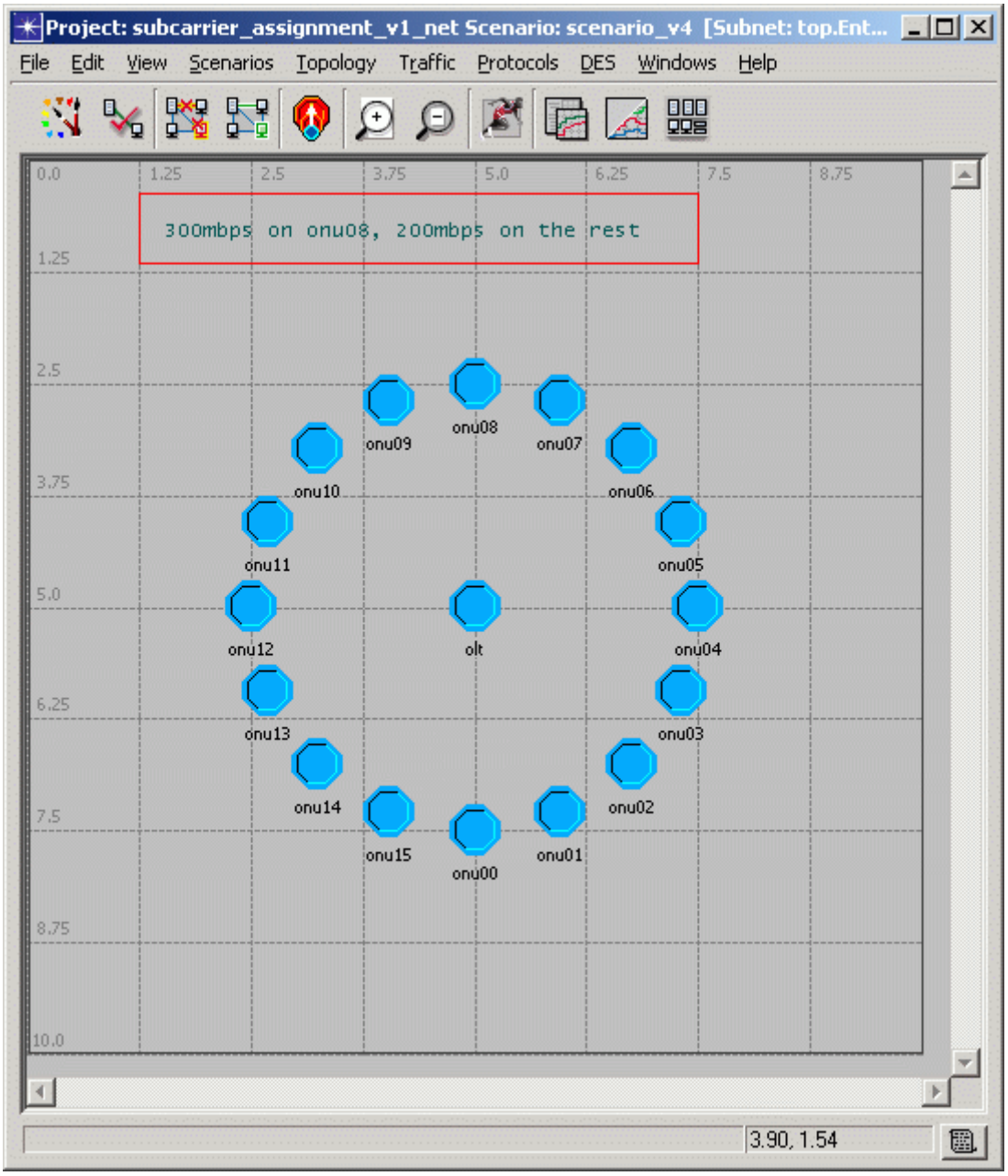

Figure A.2: Network model

At first we will focus on the OLT, a figure of the OLT on the node level is shown in figure 3. It consists of a single processing module called proc.

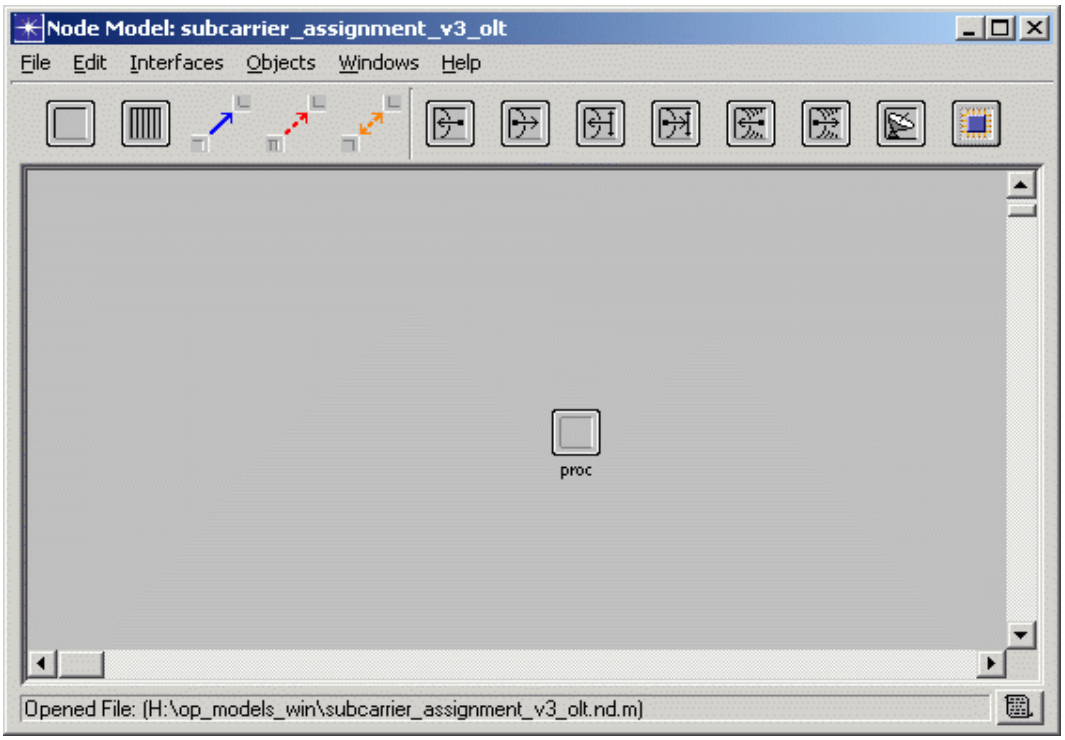

Figure A.3: OLT at node level

Looking at the process level, shown in figure 4, there is an init state and an idle state.

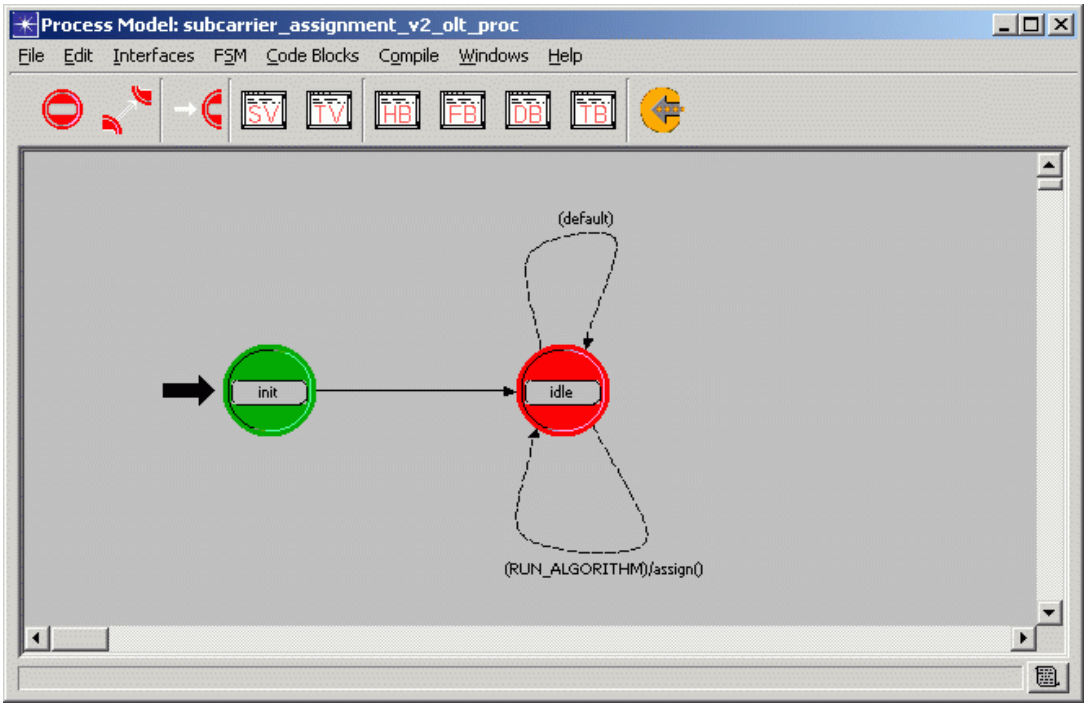

Figure A.4: OLT at process level

When the simulation starts the init state is entered and is the OLT initialized. Then it moves to the idle state where it waits until the first interrupt is triggered, it continues waiting for interrupts causing it to become active once every time epoch.

Every ONU uses the same node level model, which can be seen in figure 5, it contains more modules then the OLT.

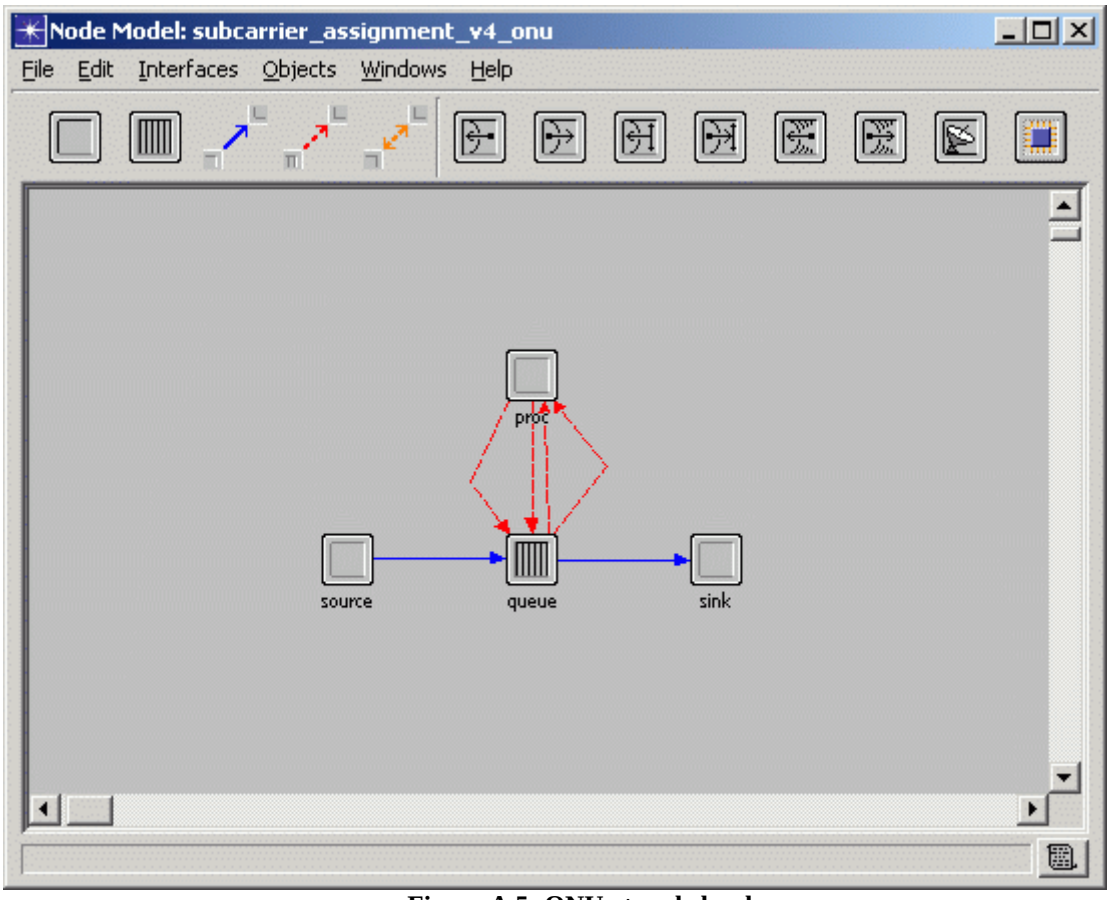

Figure A.5: ONU at node level

The proc module is used for reading assignments and controlling the queue module via the statistic wires (red dashed arrows). The source, queue and sink modules form a simple queuing system. Between those modules the packets flow via the packet streams (blue arrows). For the source and sink we use default OPNET models, some changes have been made to the default queue model though.

The default model didn't support a service rate of zero. But because we do need such functionality that possibility was added. Now when the service rate is zero, the queue accepts new packets, but doesn't process any packets. Further code was added to keep track of certain values inside the queue. This is discussed more extensively in the section on simulation output in chapter 5.3.

The state machine for proc module at the ONU is very similar to the one of the OLT, as can be seen in figure 6. It initializes in the init state at the start of the simulation and then waits in idle state to read the assignment once every time epoch.

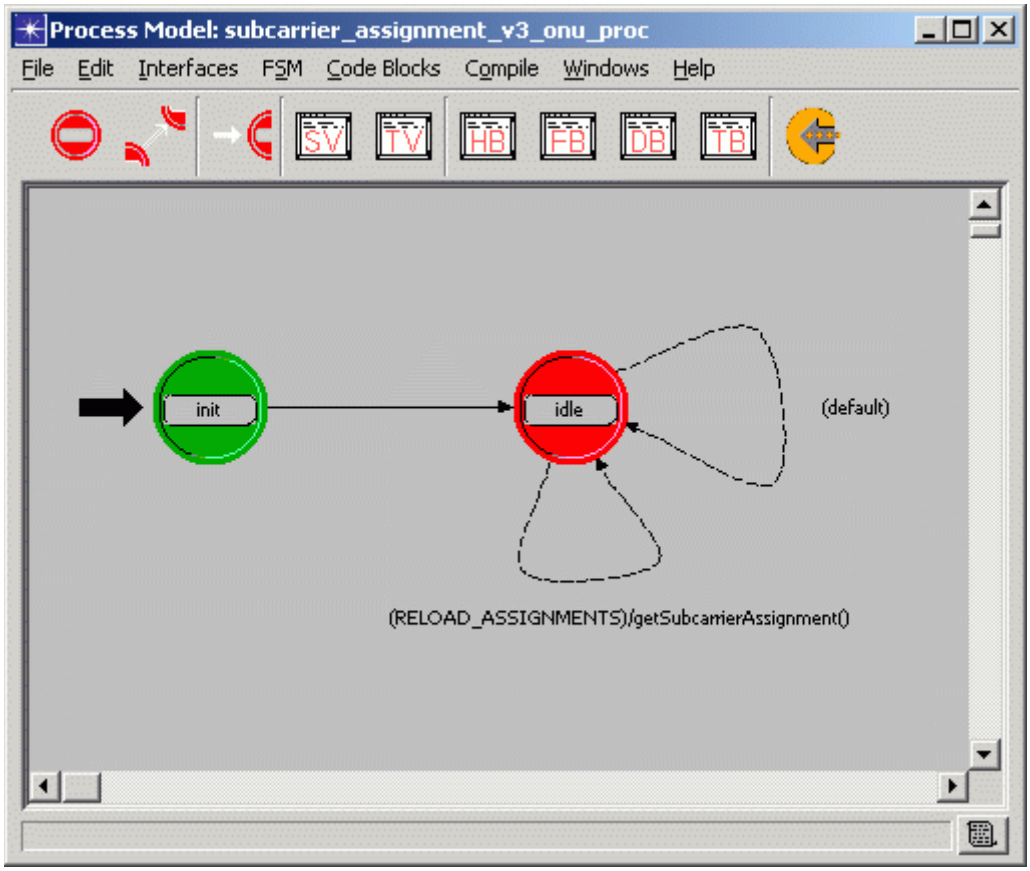

Figure A.6: The ONU at process level

Appendix B The complete results

In the simulation chapter we only presented a part of the data that was obtained from the simulations. This is because we end up with lots of data from the simulation and not all of that data is interesting to present in the main part of the thesis. But for completeness it should of course be available and that is done in this appendix.

Further was much of the data presented in charts. The reason is that charts make it easier to interpret the data. The actual values on which the charts are based are presented here.

1 Loaded bits per symbol

We start with the loaded bits per symbol values from the four main cases.

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	16.259430	13.513383	13.471004	16.714498
1	16.307311	13.617100	13.461338	16.771004
2	16.540991	13.889591	13.836803	17.007063
3	17.305725	15.053532	14.546468	17.790706
4	17.691078	13.886989	14.892193	18.330855
5	16.272119	13.318216	13.484387	16.705576
6	16.133755	13.151301	13.287360	16.510780
7	16.281908	13.306320	13.496282	16.715613
8	16.345601	13.443866	13.647584	16.823048
9	17.958315	14.873234	14.969888	18.801115
10	16.325725	13.571376	13.625279	16.779182
11	16.777670	14.140148	14.127881	17.412639
12	16.380867	13.482528	13.671747	16.918959
13	16.781413	13.866171	14.158736	17.363197
14	16.398042	13.739777	13.755390	16.928625
15	16.155985	13.415242	13.447955	16.644610

Table B.1: Average total loaded bits per symbol case 1

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	16.522007	13.513383	13.471004	16.714498
1	16.085799	13.617100	13.461338	16.771004
2	16.819033	13.889591	13.836803	17.007063
3	17.035514	15.053532	14.546468	17.790706
4	18.093358	13.886989	14.892193	18.330855
5	16.084585	13.318216	13.484387	16.705576
6	16.349889	13.151301	13.287360	16.510780
7	16.064783	13.306320	13.496282	16.715613
8	16.597745	13.443866	13.647584	16.823048

9	17.549046	14.873234	14.969888	18.801115
10	16.590087	13.571376	13.625279	16.779182
11	16.497497	14.140148	14.127881	17.412639
12	16.686691	13.482528	13.671747	16.918959
13	16.496283	13.866171	14.158736	17.363197
14	16.743618	13.739777	13.755390	16.928625
15	15.883371	13.415242	13.447955	16.644610

Table B.2: Average total loaded bits per symbol case 2

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	16.548798	14.350186	13.945353	17.068030
1	16.577001	14.271375	13.940520	17.092565
2	16.525328	14.068401	13.916729	16.957249
3	16.557819	13.918959	13.908179	17.023420
4	16.633110	13.843123	13.956506	17.104461
5	16.630037	13.732714	13.927509	17.125279
6	16.696357	13.662825	13.976952	17.200743
7	16.584040	13.806692	13.916729	17.043494
8	16.604585	13.780669	13.950929	17.123420
9	16.585626	14.004089	13.842007	17.090334
10	16.556109	14.066914	13.941636	17.102230
11	16.564808	13.799628	13.860223	17.122305
12	16.561611	13.831970	13.933457	17.091822
13	16.558340	13.676580	13.889963	17.101487
14	16.520149	13.550558	13.876208	17.119331
15	16.564610	13.632342	13.910037	17.118959

Table B.3: Average total loaded bits per symbol case 3

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	16.855936	14.350186	13.945353	17.068030
1	16.292862	14.271375	13.940520	17.092565
2	16.781958	14.068401	13.916729	16.957249
3	16.280471	13.918959	13.908179	17.023420
4	16.897125	13.843123	13.956506	17.104461
5	16.331252	13.732714	13.927509	17.125279
6	16.975886	13.662825	13.976952	17.200743
7	16.304436	13.806692	13.916729	17.043494
8	16.891549	13.780669	13.950929	17.123420
9	16.280843	14.004089	13.842007	17.090334
10	16.862974	14.066914	13.941636	17.102230
11	16.249665	13.799628	13.860223	17.122305
12	16.897621	13.831970	13.933457	17.091822
13	16.210310	13.676580	13.889963	17.101487
14	16.882751	13.550558	13.876208	17.119331
15	16.245204	13.632342	13.910037	17.118959

Table B.4: Average total loaded bits per symbol case 4

For the confidence intervals from the first four cases only stable matching results into useful results, the other methods all result in identical values for the different OPNET seeds.

ONU	Case 1	Case 2	Case 3	Case 4
0	0.011332	0.006617	0.017703	0.006146
1	0.014015	0.009974	0.010113	0.010873
2	0.013590	0.007896	0.011568	0.014198
3	0.014037	0.012158	0.011185	0.011272
4	0.018016	0.008716	0.012173	0.010230
5	0.011844	0.009567	0.012893	0.012948
6	0.012100	0.005792	0.013018	0.011750
7	0.010136	0.011195	0.013394	0.013419
8	0.013417	0.008411	0.014210	0.009112
9	0.021328	0.014843	0.014627	0.009423
10	0.013237	0.006418	0.011175	0.009479
11	0.014478	0.019491	0.008593	0.012372
12	0.015879	0.008136	0.017578	0.010831
13	0.011371	0.012072	0.020059	0.012456
14	0.013874	0.009837	0.012872	0.007931
15	0.011622	0.013249	0.013298	0.012983

Table B.5: Confidence intervals (for 95%) of loaded bits per symbol

The elasticity experiments resulted in the following loaded bits per symbol values for the stable matching method..

ONU	Case 5	Case 6	Case 7	Case 8
0	16.907138	16.927757	16.997348	17.007014
1	16.890012	16.555266	16.551722	16.545006
2	16.849343	16.528352	16.519009	16.507831
3	16.888947	16.543346	16.546592	16.546716
4	16.549120	16.619727	16.617522	16.612392
5	16.528699	16.623123	16.613110	16.609021
6	16.620942	16.696506	16.690781	16.680025
7	16.477001	16.561338	16.551177	16.555291
8	16.513680	16.585626	16.582800	16.573383
9	16.488352	16.547782	16.548699	16.557348
10	16.448005	16.517348	16.512292	16.520545
11	16.438389	16.527584	16.527906	16.515242
12	16.435514	16.506468	16.491103	16.498513
13	16.407014	16.487856	16.491623	16.485774
14	16.396406	16.467435	16.466072	16.472491
15	16.419430	16.511301	16.496555	16.504411

Table B.6: Loaded bits per symbol for elasticity experiments

The equalizing effect experiments resulted in the following loaded bits per symbol values. For the Hungarian method they are identical for all three cases.

	Stable matching			Hungarian
ONU	Case 10	Case 11	Case 12	Case 10,11,12
0	16.25943	16.2856	16.62838	16.73542
1	16.30731	16.40722	16.71787	16.91389
2	16.54099	16.53264	16.68769	16.98819
3	17.30572	17.29866	16.78972	17.82986
4	17.69108	17.67921	16.83051	18.32431
5	16.27212	16.24273	16.61764	16.67639
6	16.13375	16.1456	16.50102	16.52361
7	16.28191	16.28417	16.62481	16.71458
8	16.3456	16.34954	16.66505	16.80139
9	17.95831	17.83384	16.82815	18.78819
10	16.32572	16.34394	16.64472	16.77292
11	16.77767	16.81111	16.7594	17.37361
12	16.38087	16.49787	16.72685	17.01597
13	16.78141	16.75898	16.78056	17.32014
14	16.39804	16.49056	16.70926	16.97431
15	16.15599	16.17843	16.59537	16.56875

Table B.7: Loaded bits per symbol for equalizing experiments

2 Mean sojourn times

The mean sojourn times for the four main cases, in each case followed by the corresponding confidence intervals, for 95% certainty.

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	0.000776	0.030140	0.003409	0.000208
1	0.000646	0.026960	0.003363	0.000178
2	0.000607	0.023165	0.002917	0.000260
3	0.000702	0.012302	0.002320	0.000304
4	0.000498	0.026513	0.001565	0.000141
5	0.000612	0.031785	0.003297	0.000197
6	0.000602	0.036296	0.003396	0.000136
7	0.000601	0.036146	0.003009	0.000164
8	0.000693	0.029832	0.003112	0.000210
9	0.000444	0.015238	0.001493	0.000161
10	0.000722	0.027753	0.003209	0.000204
11	0.000828	0.021185	0.003054	0.000272
12	0.000817	0.029417	0.003181	0.000269
13	0.000568	0.023461	0.002286	0.000130

14	0.000847	0.024137	0.003316	0.000151
15	0.000789	0.030360	0.003231	0.000184

Table B.8: Mean sojourn times case 1

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	0.0000324	0.0000795	0.0000229	0.0000013
1	0.0000340	0.0000710	0.0000164	0.0000014
2	0.0000394	0.0000607	0.0000198	0.0000022
3	0.0000383	0.0000379	0.0000180	0.0000021
4	0.0000345	0.0000745	0.0000121	0.0000010
5	0.0000323	0.0001194	0.0000196	0.0000012
6	0.0000342	0.0001596	0.0000296	0.0000003
7	0.0000446	0.0001062	0.0000153	0.0000013
8	0.0000583	0.0001059	0.0000154	0.0000018
9	0.0000301	0.0000442	0.0000083	0.0000014
10	0.0000415	0.0000990	0.0000155	0.0000013
11	0.0000310	0.0000571	0.0000170	0.0000012
12	0.0000402	0.0000760	0.0000153	0.0000014
13	0.0000386	0.0000411	0.0000104	0.0000001
14	0.0000461	0.0000690	0.0000222	0.0000007
15	0.0000308	0.0000928	0.0000179	0.0000011

Table B.9: Confidence intervals mean sojourn times case 1

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	0.002023	5.625757	5.890736	0.000865
1	0.000341	0.010187	0.000324	0.000112
2	0.001943	3.795065	4.102134	0.001238
3	0.000394	0.005826	0.000488	0.000165
4	0.001267	3.783692	0.517584	0.000623
5	0.000323	0.011361	0.000400	0.000117
6	0.001324	6.964275	6.386532	0.000672
7	0.000354	0.012521	0.000340	0.000098
8	0.002055	5.870990	5.160649	0.001036
9	0.000235	0.006855	0.000257	0.000108
10	0.002002	5.530382	5.213557	0.000994
11	0.000475	0.009080	0.000444	0.000150
12	0.002088	5.805397	5.064311	0.001167
13	0.000321	0.009623	0.000305	0.000088
14	0.001941	4.799258	4.743776	0.000724
15	0.000412	0.011394	0.000331	0.000106

Table B.10: Mean sojourn times case 2

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	0.0001408	0.0344268	0.0339956	0.0000111
1	0.0000173	0.0000179	0.0000032	0.0000007

2	0.0001084	0.0209298	0.0213262	0.0000157
3	0.0000213	0.0000079	0.0000022	0.0000011
4	0.0000721	0.0263789	0.0188771	0.0000086
5	0.0000132	0.0000209	0.0000035	0.0000006
6	0.0000968	0.0297036	0.0302293	0.0000065
7	0.0000166	0.0000266	0.0000023	0.0000004
8	0.0001113	0.0296834	0.0302300	0.0000142
9	0.0000161	0.0000143	0.0000023	0.0000006
10	0.0001024	0.0275832	0.0283298	0.0000120
11	0.0000218	0.0000156	0.0000028	0.0000015
12	0.0001413	0.0268722	0.0270213	0.0000202
13	0.0000181	0.0000197	0.0000026	0.0000000
14	0.0000903	0.0287440	0.0291263	0.0000104
15	0.0000282	0.0000248	0.0000036	0.0000007

Table B.11: Confidence intervals mean sojourn times case 2

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	0.000844	0.018158	0.002962	0.000263
1	0.000657	0.020159	0.002962	0.000228
2	0.000681	0.022282	0.003078	0.000268
3	0.000674	0.022841	0.002996	0.000208
4	0.000611	0.022785	0.002883	0.000199
5	0.000596	0.025813	0.002890	0.000189
6	0.000524	0.027440	0.002389	0.000134
7	0.000753	0.023787	0.003127	0.000346
8	0.000703	0.028497	0.003146	0.000138
9	0.000745	0.022688	0.003204	0.000265
10	0.000647	0.022812	0.002641	0.000193
11	0.000722	0.025778	0.002893	0.000207
12	0.000701	0.025929	0.002731	0.000202
13	0.000780	0.028059	0.002853	0.000236
14	0.000805	0.028523	0.002810	0.000170
15	0.000804	0.029282	0.002700	0.000208

Table B.12: Mean sojourn times case 3

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	0.0000565	0.0000507	0.0000179	0.0000017
1	0.0000391	0.0000552	0.0000172	0.0000017
2	0.0000358	0.0000489	0.0000110	0.0000020
3	0.0000416	0.0000929	0.0000170	0.0000018
4	0.0000433	0.0000506	0.0000126	0.0000015
5	0.0000492	0.0000632	0.0000164	0.0000019
6	0.0000437	0.0000948	0.0000214	0.0000005
7	0.0000383	0.0000627	0.0000165	0.0000035
8	0.0000430	0.0000773	0.0000161	0.0000007

9	0.0000392	0.0000616	0.0000153	0.0000021
10	0.0000302	0.0000755	0.0000182	0.0000012
11	0.0000399	0.0000599	0.0000155	0.0000014
12	0.0000518	0.0000751	0.0000172	0.0000012
13	0.0000476	0.0000698	0.0000165	0.0000016
14	0.0000520	0.0000858	0.0000199	0.0000011
15	0.0000428	0.0000899	0.0000124	0.0000015

Table B.13: Confidence intervals mean sojourn times case 3

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	0.002398	2.576236	3.976102	0.001309
1	0.000383	0.008419	0.000482	0.000140
2	0.002268	3.984262	4.042911	0.001433
3	0.000431	0.008899	0.000483	0.000112
4	0.001910	4.266810	3.923695	0.001148
5	0.000347	0.009607	0.000390	0.000111
6	0.001495	4.793219	3.769039	0.000645
7	0.000422	0.009596	0.000455	0.000192
8	0.001848	4.384644	3.784025	0.000708
9	0.000400	0.009371	0.000490	0.000148
10	0.001593	3.694374	4.035367	0.000898
11	0.000404	0.010339	0.000375	0.000117
12	0.002059	4.493368	4.069455	0.001092
13	0.000448	0.011048	0.000382	0.000138
14	0.002166	5.499378	4.276317	0.000912
15	0.000419	0.011134	0.000372	0.000117

Table B.14: Mean sojourn times case 4

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	0.0001139	0.0338321	0.0349525	0.0000221
1	0.0000186	0.0000174	0.0000032	0.0000010
2	0.0000905	0.0215877	0.0214861	0.0000249
3	0.0000145	0.0000127	0.0000044	0.0000010
4	0.0001561	0.0270083	0.0277847	0.0000137
5	0.0000160	0.0000165	0.0000038	0.0000006
6	0.0001273	0.0303436	0.0312713	0.0000070
7	0.0000161	0.0000187	0.0000037	0.0000016
8	0.0001124	0.0300463	0.0305668	0.0000109
9	0.0000189	0.0000150	0.0000028	0.0000009
10	0.0001200	0.0269490	0.0283559	0.0000116
11	0.0000233	0.0000170	0.0000030	0.0000007
12	0.0000666	0.0274228	0.0271282	0.0000170
13	0.0000225	0.0000211	0.0000029	0.0000006
14	0.0001251	0.0291140	0.0290806	0.0000144
15	0.0000228	0.0000245	0.0000026	0.0000006

Table B.15: Confidence intervals mean sojourn times case 4

Here are the complete results for case 6, 7 and 8, used in exploring the effect of overloading on mean sojourn times.

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	0.001841	2.575485	3.976168	0.001307
1	0.000207	0.008415	0.000483	0.000139
2	0.000219	0.008947	0.000469	0.000161
3	0.000237	0.008912	0.000483	0.000112
4	0.000188	0.008972	0.000422	0.000106
5	0.000163	0.009600	0.000392	0.000111
6	0.000149	0.010533	0.000322	0.000089
7	0.000249	0.009607	0.000453	0.000192
8	0.000229	0.011388	0.000408	0.000089
9	0.000241	0.009381	0.000490	0.000149
10	0.000202	0.009316	0.000380	0.000117
11	0.000238	0.010355	0.000370	0.000115
12	0.000236	0.010155	0.000412	0.000117
13	0.000259	0.011055	0.000381	0.000138
14	0.000243	0.010476	0.000368	0.000092
15	0.000242	0.011141	0.000374	0.000117

Table_Apx B.16: Mean sojourn times for case 6

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	8.320926	15.893182	16.967304	8.129310
1	0.000212	0.008403	0.000484	0.000140
2	0.000231	0.008970	0.000471	0.000162
3	0.000235	0.008903	0.000486	0.000112
4	0.000207	0.008968	0.000424	0.000107
5	0.000167	0.009591	0.000391	0.000111
6	0.000157	0.010511	0.000320	0.000089
7	0.000250	0.009601	0.000457	0.000194
8	0.000206	0.011377	0.000412	0.000089
9	0.000245	0.009362	0.000490	0.000149
10	0.000199	0.009315	0.000378	0.000118
11	0.000240	0.010351	0.000371	0.000116
12	0.000239	0.010141	0.000409	0.000117
13	0.000259	0.011084	0.000385	0.000139
14	0.000253	0.010455	0.000372	0.000091
15	0.000250	0.011148	0.000374	0.000117

Table B.17: Mean sojourn times for case 7

ONU	Stable matching	Contiguous	Interleaved	Hungarian
0	17.814483	23.913668	24.768948	17.676220
1	0.000209	0.008413	0.000483	0.000139
2	0.000234	0.008951	0.000465	0.000162
3	0.000230	0.008895	0.000485	0.000112
4	0.000212	0.008968	0.000422	0.000106
5	0.000172	0.009595	0.000387	0.000110
6	0.000156	0.010527	0.000320	0.000089
7	0.000253	0.009586	0.000457	0.000192
8	0.000215	0.011394	0.000409	0.000089
9	0.000241	0.009366	0.000490	0.000149
10	0.000200	0.009312	0.000379	0.000117
11	0.000241	0.010357	0.000371	0.000116
12	0.000229	0.010152	0.000412	0.000117
13	0.000243	0.011047	0.000383	0.000139
14	0.000242	0.010477	0.000369	0.000091
15	0.000240	0.011120	0.000372	0.000117

Table B.18: Mean sojourn times for case 8