Algorithms for Task Sharing in the Internet of Things

by

Daniel Connaughton

Supervisor: Dr. Georgios Iosifidis

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science & Statistics

A dissertation submitted to the University of Dublin, in partial fulfilment of the requirements for the degree of

Master in Computer Science (MCS)

Submitted to the University of Dublin, Trinity College, May, 2018

Declaration

I, Daniel Connaughton, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Signature: 

Date: 25/05/2018

Summary

The Internet of Things is emerging as the next wave of the Internet. Although the Internet has always been made up of things, nowadays almost any object with sensing, actuating or computational capabilities can be connected to the Internet. However, the Internet of Things is more than just connecting mass amounts devices to the Internet. A promising feature of the devices used in an Internet of Things network, is their ability to cooperate with one another to perform a task and improve their performance. In this dissertation we design policies that allow these devices to successfully cooperate to exchange resources such as energy or wireless bandwidth with one another to complete tasks.

In this dissertation we adapt the Max-Weight Scheduling policy to determine how nodes should cooperate. The Max-Weight policy has a number of benefits that make it desirable as a scheduling policy. The first is that it is a distributed algorithm so it does not require the need for a centralised node to organise coordination. The second is that it does not require a priori knowledge about the production and request rates of nodes to determine who should be served. It is also throughput optimal in that it has the ability to stabilise a queueing network whenever possible. Stability in a resource sharing network is crucial as it guarantees that nodes are not starved of the resources that they require in order to sustain existence.

We then build on top of the Max-Weight policy and extend to the case where there are a number of different types of resources that nodes can exchange among one another. Each producer can do some planning and decide how to use the resources available to them best serve the consumers with the largest demands. For example, if we consider a sensor that is used to measure air quality in an urban environment, the sensor can decide how much energy it should use to either record measurements or to help other nodes by exchanging measurements with them.

After investigating the Max-Weight policy in a number of different topologies we found a Ring or a fully connected Mesh network to be the most stable. However, we also found that the Max-Weight policy also causes a lot of resources to be wasted when deployed in highly dense Mesh networks. To avoid the number of resources wasted, the Average-Weight policy is presented. As the Average-Weight policy wastes far less resources than

the Max-Weight policy it results in much lower queue sizes. Smaller queue sizes mean that requests spend less time in a network waiting to be serviced. Most importantly the Average-Weight policy retains the three properties that the Max-Weight policy offers. It is distributed, it does not require any a priori knowledge about nodes and so far it has been able to stabilise any network which the Max-Weight policy stabilised.

Production scheduling also proved to be very key in terms of ensuring stability in networks that exchange more than one resource. Whereas self-interested nodes that deploy a production policy that only reflects their own needs or a policy that results in nodes only producing low-cost plans were shown to lead to the demise of the network in which they are a part of.

To conclude, we found both the Max-Weight and the Average-Weight policies to be two different ways to ensure stability in networks that exchange resources. However, the Average-Weight policy wastes far less resources than the Max-Weight policy. Producers scheduling the production of resources is also a key factor in ensuring network stability.

Abstract

Algorithms for Task Sharing in the Internet of Things

By Daniel Connaughton

Supervisor: Dr. Georgios Iosifidis

Master in Computer Science (MCS), 2018

The Internet of Things, (IoT), can be viewed as a network of IoT devices embedded in physical objects that we use every day. An IoT device is any device that has any sensing, actuating and/or computational capabilities. A promising feature of IoT devices is their ability to cooperate with one another by exchanging resources to improve their performance.

In this dissertation, we will design and analyse policies that enable IoT devices to exchange resources. Developing resource exchange policies in IoT networks is a challenging and very important problem. To tackle this problem, we will leverage the celebrated Max-Weight algorithm which was initially devised (in 1993) for optimizing packet routing in multi-hop data networks and since then has been applied to a number of problems in a variety of areas including communication networks, computing systems, transportation networks, economics, and so on.

We will adapt this algorithm to the IoT resource sharing context for deciding how each producer allocates their resources to different consumers. Furthermore, we will consider the increasingly relevant scenario where there are a number of different resource types that nodes exchange and producers can schedule the production of their resources to generate different types of resources to best serve the neediest consumers. We will study what the necessary conditions for these devices to successfully collaborative and achieve a sustainable network are.

The Max-Weight policy when deployed in different network topologies will be evaluated and based on results new task sharing algorithms will be designed. Different production policies will also be analysed and their advantages and disadvantages will be presented.

Acknowledgements

I would like to take this opportunity to thank Dr. George Iosifidis and Victor Valls for all their help and the resources and material they provided for this dissertation.

I would also like to thank my family for their support over these last few years.

Table of Contents

Table of Figures

Table of Tables

Table of Equations

1 Introduction

1.1 Introduction

This dissertation is about designing task sharing algorithms for Internet of Things (IoT) networks. Tasking sharing can be thought of as dividing a large problem into smaller tasks so that it can be solved more optimally and efficiently [1]. Task sharing in an IoT network involves coordinating a number of IoT devices so they can cooperate with one another by exchanging different resources to complete tasks.

1.2 What are Internet of Things Networks?

The expansion of embedded systems has led to the development of the IoT [2]. The IoT can then be viewed as a network of many different IoT devices that collect and exchange information among one another with an aim to extend the availability of the Internet to anyplace at any time [3]. IoT networks are expected to have a profound impact on both its users and their environment. Mainly due to the wide range of areas that IoT applications are expected to be present in and the substantial number of IoT devices that are expected to be a part of the IoT. An IoT device is any device that has any sensing, actuating and/or computational capabilities. These devices are then equipped with identifiers and wireless connectivity so that they can exchange information automatically between each other in real time [3]. Examples of IoT applications can be found in manufacturing, transportation, smart cities and telecommunications [4].

An example of an IoT application can be found in a smart home. A smart home consists of many IoT sensors and actuators including light, temperature and motion sensors. A smart home also has a computational unit. The computational unit is used to reason about recordings made by the different sensors and make changes to the environment to adapt to these recordings. For example, the opening of a window in a room when the temperature sensors record a reading above some predefined threshold. IoT devices offer several benefits when deployed in a smart home including reducing the amount of energy that is wasted, reducing the monetary costs of running a home and reducing the risks of exposure to harmful air pollutants [5].

IoT is expected to have a huge impact in any area that deals with energy management. In [6], an example of how IoT technologies are being used in the National

Grid in the U.K. to provide a healthy maintainable electrical network is presented. The authors of [5] list some of the benefits of using IoT applications for electronic vehicle charging such as the reliability, resiliency, adaptability, and energy efficiency. But such benefits seem to be a recurring theme provided by IoT devices in most IoT applications. More examples of IoT devices include smart phones, autonomous vehicles, traffic light systems, robotics in manufacturing and the devices that have always been a part of the internet.

A promising feature of devices used in IoT applications is their ability to cooperate with one another to perform a task by exchanging resources, such as wireless bandwidth or processing power, to improve their performance by consuming less power or decreasing the execution time of a task. There are two ways in which devices can cooperate. In the first scenario, a device executes a task on behalf of another device that the second device cannot execute itself but it requires its output. In the second scenario, one device exchanges some of its own resources with another device so that the second device may perform a task.

1.3 What are Challenges for Internet of Things Networks?

There are a number of different challenges for IoT devices and networks. A lot of IoT devices are often low powered battery-operated devices with limited computational capabilities and as a result it is a requirement for them to consumer as little power as possible [7]. Another important challenge is, due to the wide range of IoT applications available and the different types of sensors and devices that are used in different applications, IoT networks can become quite diverse. Also, because IoT networks can have nodes dynamically joining and leaving networks at any time, task sharing in IoT networks is not a straightforward problem. As a result of these challenges, there is need of dynamic policies that can be followed in order for devices to successfully collaborate to exchange resources over a long period of time.

1.4 Research Aims

The aim of this dissertation is to study the policies from “Dynamic Policies for Cooperative Networked Systems” [8]. The policies will then be used to design and analyse new algorithms that enable task sharing among nodes in IoT networks. The policies will be implemented and simulated on a number of different networks using Python and iGraph (which is an open source software package used for network analysis).

There are two aspects of the paper mentioned above presented in this dissertation. The first is the Max-Weight policy which was initially devised (in 1993) for optimizing packet routing in multi-hop data networks [9]. Since then has been applied in a variety of problems in communication networks, computing systems, transportation networks, economics, and so on. There are a number of benefits that the Max-Weight policy offers as a scheduling policy. The Max-Weight policy is throughput optimal which means it will stabilise a queueing network whenever possible [10]. The Max-Weight policy does not require any a priori information about the demand or resource generation rates about nodes when determining who to serve. The Max-Weight policy is also a distributed algorithm as it does not require a central node to organise coordination [8]. Under the Max-Weight policy each producer of resources simply selects the consumer with the highest number of demands that it is connected to. The Max-Weight policy will be adapted to the context of IoT resource sharing where producers of resources decide which consumers they should allocate their resources to.

Different network topologies such as Mesh, Ring and Star will be considered. How the Max-Weight policy is affected by different topologies will be evaluated. The best and worst performing topologies in terms of network stability will be further evaluated. Different network properties and characteristics such as assortativity mixing and node degree will also be assessed and their impact will be discussed.

The second aspect is the increasingly relevant scenario of when there are a number of resources that nodes exchange. Producers can schedule the production of the resources available to them into different types of resources to best serve the consumers with the highest requests. Producers can put more effort into generating one type of resource at the expense of reducing the amount of effort put into generating other types of resources in an attempt to best serve the consumers in the network with the highest demands. One way to think of production planning is to imagine in a smart home a solar panel produces solar renewable energy. The renewable energy can then be converted into electricity or into hot water depending on the needs of the smart home [11]. A number of different production scheduling algorithms will be compared and contrasted to view the benefits of using production scheduling. More details about this paper and these policies will be discussed further in the next chapter.

1.5 Methodology

A number of different types of networks will be modelled using Python and iGraph. Each different network will be measured based on evaluation criteria which includes the conditions for sustainability [8] and the strong stability requirement for queues [12]. Stability is an important metric in the context of resource sharing as if a network is stable it means that nodes are not starved of the resources they require to sustain their existence over a period of time. Both the best and worst performing topologies will be studied to see what makes them more or less suited than others for the Max-Weight policy. The advantages of using the Max-Weight policy over other node selection algorithms will be discussed. Based on results new task sharing algorithms will be designed and their advantages and disadvantages over using the Max-Weight policy will be discussed. Then the benefits of producers scheduling the production of their resources over other production algorithms will be compared. Finally, both the policies for selecting nodes and for production scheduling will be applied to hypothetical but real-life network examples.

1.6 Contribution

A number of contributions will be made by this dissertation including,

- Showing that the most stable topologies for the Max-Weight policy are a fully connected Mesh network and a Ring topology.
- Showing how the Max-Weight policy can cause resources to be wasted in certain situations.
- Designing a new task sharing algorithm to avoid wasting resources.
- Showing how production scheduling can increase network stability when the number of types of resources in the network is more than one.
- Describing the impact that graph properties such as assortative mixing have on network stability.
- Applying the Max-Weight policy and production planning to a social network graph and a power grid graphs.

1.7 Outline

The rest of this dissertation is organized as follows. In the State of the Art chapter a background of the Internet of Things and how it is expected to impact different areas is discussed. Then a detailed description of the “Dynamic Policies for Cooperative Networked

Systems" paper, which is used as the system model for this dissertation, is provided. In chapter three, how different topologies impact the Max-Weight policy and other node selection algorithms will be presented. Based on results a new task sharing algorithm will be designed. In the fourth chapter the idea of producers scheduling the production of their resources will be presented. Then the advantages and disadvantages of production scheduling and other production scheduling algorithms will be discussed. In chapter five different graph properties and their effect on network stability will be examined. Finally, in chapter six a number of examples of using the Max-Weight policy and production scheduling will be presented.

2 State of the Art

2.1 The Internet of Things

The Internet of Things can be viewed as a network of computing devices embedded in physical objects that we use every day. These devices can be any internet enabled device that has any form of sensing, actuating and/or computational capabilities which enables them to connect to and communicate with one another in real-time to perform a wide range of tasks [3]. Such devices can range from a smart phone, to robotics used in manufacturing, to the high-performance computers that have always been a part of the Internet [2]. It is estimated that the IoT could be composed of nearly thirty billion devices by 2020 [6]. By 2025, IoT devices may even include food packaging and furniture which further highlights the vast number of devices that are expected to be connected to the IoT [13].

IoT applications have the ability to automate everyday functions by placing a small computer inside of an object so that it can receive and exchange information and resources in real-time which is one of the key aspects driving this new technology revolution [7], [3]. An example of an IoT application can be found in oil and gas exploration where sensors on an oil pipeline monitor changes in pressure. When the sensors register a pressure change above some threshold the pumps shut down in real time to avoid any disaster [4]. Another example can be found in smart cities, where smart meters are used to measure electricity usage in a smart home in real-time. The data is collected and sent back to the provider where supply levels are adjusted appropriately so that resources are not wasted [14]. Further examples of IoT applications can be found in Figure 2.1 on the following page.

2.2 Related Work

As the Internet of Things is a relatively new paradigm there are no standards on how task sharing should be implemented in IoT networks. Instead the algorithms described in this dissertation are based on the policies from [8] which use elements of Graph Theory and Queueing Theory to approach task sharing. The policies are further described in the following sections.

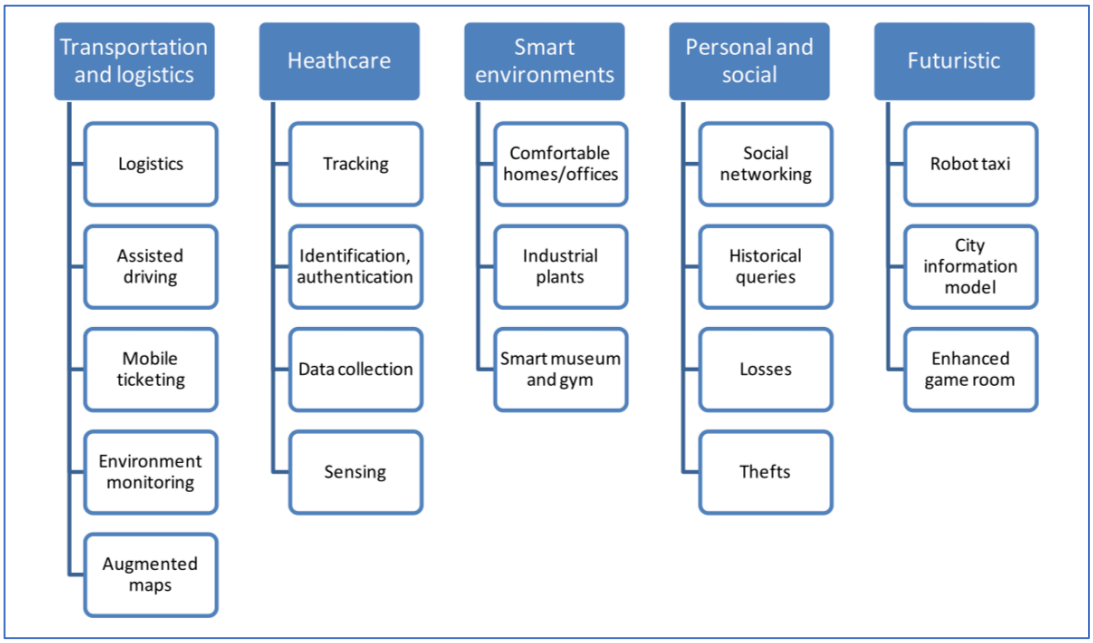

Figure 2.1 - IoT Applications.

Examples of the different areas that IoT applications are expected to have an impact in [13].

2.2.1 Dynamic Policies for Cooperative Networks

The proposed policies are not directly aimed at IoT, they are instead aimed at Economic Entities (EE) which can be any individual or organisation that makes requests for resources in order to sustain its existence. Each EE makes requests for resources but can also generate resources to satisfy those requests. Therefore, each EE has two roles. It acts as both a producer and a consumer of resources. For example, consider an IoT sensor that is based on an energy harvesting mechanism such as a solar panel. The solar panel generates electricity but it also might consumer some energy in the process and in other operations.

An EE is self-sustainable if the rate at which it produces resources is greater than the rate at which it makes requests for resources. If the rate that resource requests are generated exceeds the rate which an entity can generate resources to satisfy those requests for a given resource type then that entity may become unsustainable. However, if there is a group or a network of entities connected to one another, then it is possible that although some entities may not be sustainable on their own for each type of resource, it may exchange resources with other entities in the network and help cover its own lack of that resource. If an exchange policy exists such that all entities in the network become sustainable then that network is also sustainable.

2.2.2 Resource Sharing and the Max-Weight Policy

The authors first cover the case where there is only one type of resource, K, that entities exchange. The entities are connected in a directed graph $G = (N, \mathcal{E})$ meaning all nodes are connected to each other by the set of edges $\mathcal{E}$ which are not necessarily bidirectional. As already mentioned, each entity in the network has two roles, it is both a producer and a consumer. Therefore, there are $j = 1, \dots, N_i$ producers who can serve consumer i and there are $i = 1, \dots, N_j$ consumers who can receive resources from producer j in the network. We assume a time slotted operation where a time slot could be a minute an hour or a day depending on the environment. At the start of each time slot each producer j in the network produces $0 \leq B_j \leq B_{max}$ units of resources. Where B_{max} is the max production rate of the network. At the start of each time slot each each consumer i generates $0 \leq A_i \leq A_{max}$ requests for resources. Where A_{max} is the max arrival rate of the network. At the end of each time slot t a consumer has $X_i(t)$ pending requests.

During each time slot t a consumer can be served by many producers but a producer may only serve one consumer. The control action of this system is to decide which producers serve each consumer during each time slot. Let $I_{ji}(t)$ be a $N\ x\ N$ binary matrix where each entry (j, i) denotes whether producer j is serving consumer i during time slot t. Each entry (j, i) can only be equal to 1 if $(j, i) \in \mathcal{E}$. Note that $(i, i) \in \mathcal{E}$ as each entity can serve its own requests.

Each producer j uses the Max-Weight policy to choose which consumer i it will serve in each time slot t. To select which consumer i to serve, each producer j simply observes the pending requests $X_i(t)$ for every consumer it is connected to and then assigns its resources to the consumer with the largest demands. Each consumer i then updates its pending requests X_i. First it subtracts the total number of resources it has received $M_i(t)$ from each producer in the current time slot t from its pending requests $X_i(t-1)$ in the previous time slot. Where $M_i(t) = \sum_{j \in N_i} I_{ji}(t) B_j(t)$. Then it adds the number of new resource requests $A_i(t)$ it has generated in that time slot to its number of pending requests to get its updated number of pending requests $X_i(t)$ for that time slot t. Pseudocode of the Max-Weight serving policy is provided in Figure 2.2 on the following page.

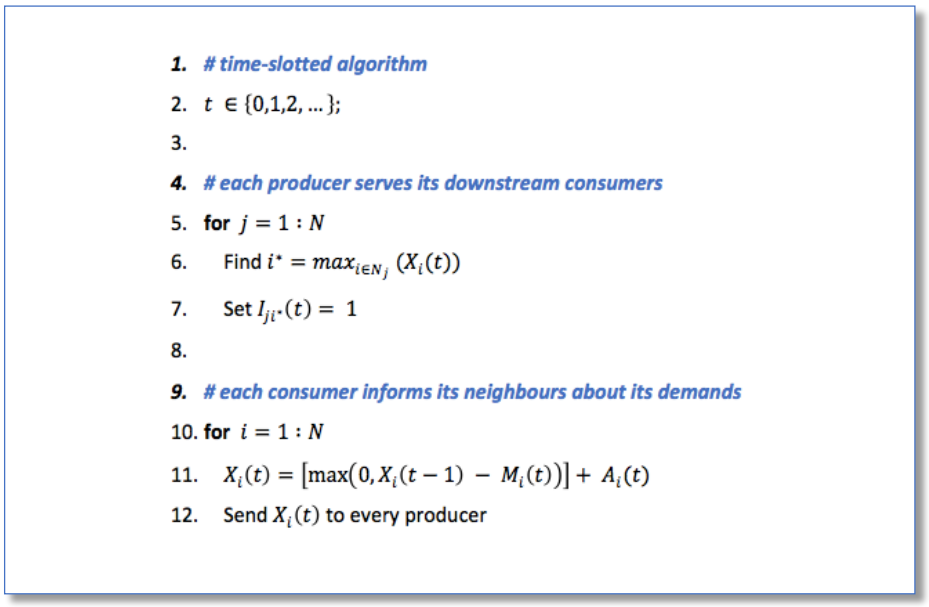

Figure 2.2 - The Max-Weight Scheduling Policy.

2.2.3 Production Planning

The authors then build on top of the Max-Weight policy and extend to the case where there is a number of different resource types that entities exchange among one another. Each consumer i now generates $A_{ik}(t)$ requests for resource k during each time slot t. Each producer j is now allowed to do some planning to decide how to best split its available resources to produce different resource types. For example, if producer j had two units of resource available it could producer either two units of resource type A or two units of resource type B or one unit of each. Producers can increase the production of a certain resource type by putting more effort into producing that type of resource at the expense of reducing its efforts spent producing other types of resources. The production choices of a producer j are represented by the set of different possible production plans P_j. Under each plan p a producer produces $B_{jk}^{p}(t)$ units of resource k during time slot t. Again, each consumer i is allowed to be served by multiple producers. However, each producer j is now allowed to serve more than one consumer as long as it only serves at most one consumer for each resource k. As before each producer j chooses the consumer i with the largest pending requests $X_{ik}(t)$ to serve for each resource type in each time slot t. After each producer j chooses which consumers to serve for each resource type k, it then selects the production plan p that will best serve those consumers. Finally, each consumer i updates its pending requests X_{ik} for each resource type k in a similar fashion as before.

2.3 Abstraction

The polices and system described in the previous two sections will be used to model IoT networks and devices. Each node can be thought of as an IoT device. Each IoT device can either generate or produce some form of a resource such as energy or wireless bandwidth depending on the device. The production and arrival rates of devices will be simulated using random number generators. Each edge in the network graphs represents a one-hop neighbour for an IoT device that it can exchange its resources with.

3 Network Topologies

3.1 Introduction

Nodes in a network graph, may only exchange resources between one another if they are connected. Two nodes in a graph G are connected if there is an edge between them. The edges of a network define the topology of that network. In the following chapter the performance of the Max-Weight policy when deployed in different network topologies will be discussed. Mainly, this chapter will look at how network stability is affected when the Max-Weight policy is used in different topologies. Network stability will be measured using the strong stability requirement (Equation 3.1) [8]. The strong stability requirement states that a queue is strongly stable if it does not grow to infinity over time. A network is strongly stable if all individual queues in that network are also strongly stable [3]. Or in other words a network is stable if the number of pending requests that each consumer has should not continually grow to infinity over. When more nodes connect to one another the number of pending requests that each node has should not continually grow to infinity over time. Also the average number of pending requests for the entire network should also not continually grow to infinity over time as average vertex degree of that network is increased.

$$\lim_{t\to\infty} sup \frac{1}{t} \sum_{T=0}^{t-1} \sum_{i=1}^{N} \mathbb{E}[X_i(t)] < \infty.$$

Equation 3.1 - The Strong Stability Requirement for Demands.

The main topology that will be studied in this chapter is a Mesh topology but Ring and Star topologies will also be discussed. Different Mesh networks will be evaluated ranging from sparse networks with little or no edges to fully connected ones. The Mesh networks will be simulated using geometric random graphs, (GRG). One way of increasing or decreasing the number of edges between nodes in a GRG is by changing the radius r of the GRG. As the radius r of a graph increases more edges are added to the graph and more nodes become connected to each other.

To test this a number of different Mesh networks were initialised using the geometric random graph function in the iGraph package. GRG are similar to Mesh networks where nodes connect to as many nodes nearby as possible. Each graph was then

passed as a parameter, a graph radius which determined the edges of the graph. When the radius $r = 0$, each node is only connected to itself and cannot exchange resources with other nodes. As the radius increases more nodes become connected until eventually the network graph becomes fully connected. A fully connected network is a network where all nodes are directly connected to each other [15]. It isn't till the radius $r \geq 0.25$ that every node in the network becomes connected to at least one other node and every node can start to make use of the Max-Weight policy.

3.2 Mesh Network Graphs

The first case studied was when there was only one type of resource $k \in K$ that nodes in a network exchange among one another. To keep the results obtained from each simulation based only on the effects of different topologies, every network was initialised with arrival rate $A_i \leq 1$ and production rate $B_j \leq 1$. Meaning each consumer i generates $0 - 1$ resource requests and each producer j generates $0 - 1$ unit of resource during each time slot. In general, when both the arrival rate and production rate are increased the number of pending requests also increases, but the networks do not necessarily become unstable. The only variables altered between simulations were the graph radius r and the number of nodes N in the network. In each simulation of the Max-Weight policy the number of time slots $t = 1000$.

Each different radius was tested on fifty different networks to calculate the average for each different radius. It appeared as though that after fifty simulations of each radius the queues were beginning to converge for these particular simulations. However, when the number of resource types increased or the number of nodes in each network were increased and other variables were tested the number of simulations for each different network was greatly increased to gain more accurate results. Networks were mainly initialised with either twenty or fifty nodes. Other networks sizes both smaller and larger were tested. For each of those network sizes a range of graph radii between $0 - 2$ was tested.

When the graph radius $r = 0$, each node is only connected to itself. That means each node can only serve its own requests and it cannot make use of the Max-Weight policy. As expected the number of pending requests grows to infinity. Although some nodes have little or no pending requests at all, there are a lot of nodes with a high number of pending requests. The nodes with the high number of pending requests also have unstable queues and as a result the network also becomes unstable, as can be seen in

Figure 3.1 below. When the radius $r = 0$ the average number of pending requests in each network is higher than when the radius $r > 0$ for almost all network sizes tested when the number of resource types $K = 1$. The queues are most unstable when the radius $r = 0$. As in they grow to infinity almost straightaway.

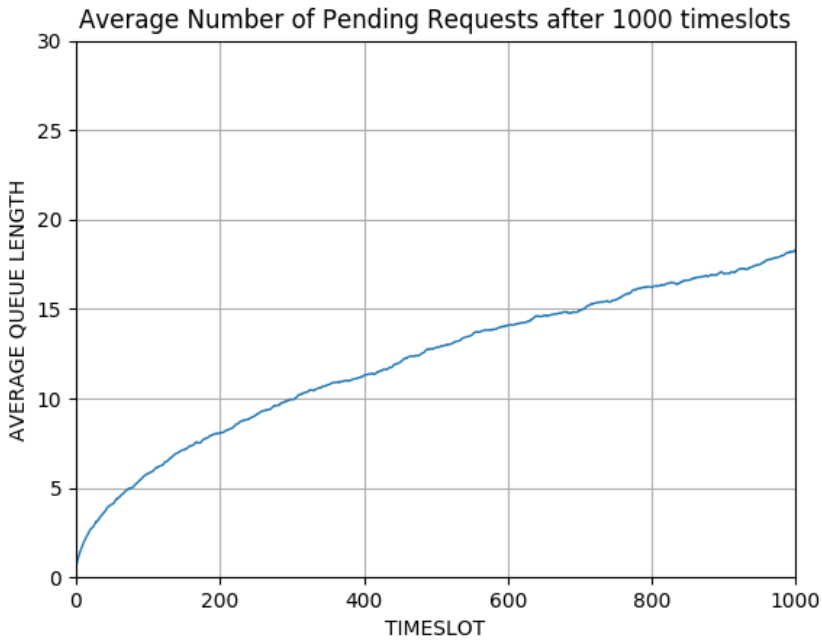

Figure 3.1 - Unstable Sparse Mesh Network.

As the radius increases more nodes become connected and can make use of the Max-Weight policy to exchange resources with each other. As a result of this the queues start to stabilise. When the radius $r = 2$ the network becomes full connected and the queues become stable. However, a fully connected network did not necessarily lead to a decrease in queue size which was original expected. When the number of nodes was equal to twenty, a fully connected network had a higher average number of pending requests than a network with twenty nodes and radius $r = 0.25$ (which is about an average vertex degree of four for that particle network size) on average. However, although the queue size for the network graph with radius $r = 0.25$ was smaller, it was not stable. When the number of nodes increased to fifty, the network graph with radius $r = 0.25$ also had a lower average number of pending requests than a fully connected graph, but it was not stable. When the number of nodes increased to 100 the results were similar.

It is also worth noting that when the number of nodes in a network increases the queues become more stable. This is mainly due to the number of nodes a nodes is

connected to increasing. Although, the increase in nodes did lead to an increase in the average number of pending requests in a fully connected network. In a network graph with a radius $r = 0.25$ there was also an increase in queue size but only slightly. The increase in nodes in a network with radius $r = 0.25$ did also improve the stability but only slightly as it was still effected by the nodes with a low number of neighbours. It is important to note that a lower queue size does not necessarily lead to a more stable network. Maybe it is more important to note that an increase in queue size does not necessarily lead to an unstable network.

3.3 Wasting Resources

When the number of resource types $k > 1$ the results are almost identical. As the radius increases the average number of pending requests also increases but the queues also become more stable. The increase in the queue size when the radius is increased is believed to be due to less nodes in a network being served as a result of the increase. For example, in a fully connected network when each producer j selects a consumer i to serve, every producer j sees the same set of consumers N_j and as a result sees the same consumer i with the highest number of pending requests X_i. This means that in a fully connected network only one consumer will be served in each time slot. Whereas in networks that have a lower average vertex degree more consumers are served which allows resources to be more evenly distributed throughout the network. As a result of more consumers being served in each time slot the queue size decreases. But as already mentioned a decrease in queue size does not necessarily mean a more sustainable network.

In the fully connected network, a number of resources are wasted because of the fact that only one consumer is served in each time slot. It is not only in fully connected Mesh networks that resources are wasted. As the number of edges increases in a Mesh network, or in other words that Mesh network becomes more dense, more and more resources are wasted. Most resources wasted in the simulations occurs in the first few time slots when each consumer might only have a relatively low number of requests. Resources are wasted in the networks with a low graph radius too but the number of resources wasted in the fully connected network is far greater. For example, imagine a fully connected network with twenty nodes and each consumer i has an arrival rate $A_i \leq 1$ and each producer j has a production rate $B_j \leq 1$. In the first time slot the max number of pending requests $X_i = 1$. There might be a number of nodes with pending requests

$X_i = 1$. However if all twenty producers, each have one resource unit to give to a consumer, selects the same consumer i to serve then there will be 19 resources wasted. This trend can continue for the first few time slots which causes a lot of resources to be wasted until the number of pending requests at each consumer increases and then less resources are wasted. In this example the number of resources types $K = 1$. But if the number of resources $K = 2$, then are is a chance that this situation can occur for each resource in the network. So instead of having 19 resources wasted there would be 38. If we increase K then even more resources could be wasted. In Figure 3.2 below the total number of resources wasted can be seen for two different graph radius. On the left the graph radius $r = 0.25$ and on the right the graph radius $r = 2$.

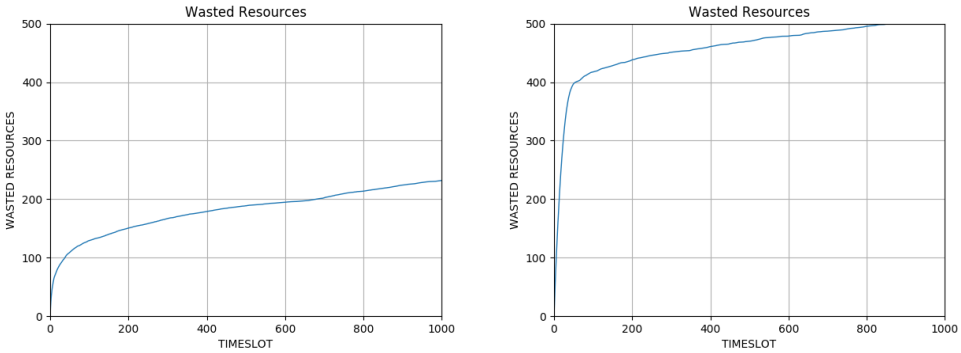

Figure 3.2 - Wasted Resources in Mesh Networks.

Although both the fully connected and sparse networks waste resources, because the fully connected network wastes so many resources when the number of pending requests in the network are relatively low it causes the number of pending requests to be much higher than the networks with a low graph radius such as $r = 0.25$. It can be seen in the graphs in the Figure 3.2 above that both the networks continue to waste resources at a similar rate. Because the sparse networks have more consumers being served in each time slot it allows the resources to be more distributed which results in less of a chance of resources being wasted. The fully connected networks are more stable than the sparse networks though. This is because in a sparse network, which results in nodes having a low number of neighbours, the consumers with the largest requests might not be connected to enough producers to receive enough resources to keep it from becoming unstable.

When the number of nodes increases it has a huge impact on the queue size. In a way it is expected as more consumers are now generating more requests. But there should also be more producers serving those requests which should keep the queue sizes the same

regardless of the network size. When the number of resource types $K = 1$ there is also an increase in the queue size as the number of nodes increased but not nearly as large as when K is increased to two. When the number of nodes increases from twenty to fifty the queue size almost doubles for a network with a radius of 0.25. But the queue size increases even further when the network becomes fully connected. This is due to even more resources being wasted when the number of nodes increases. It has already been shown that the Max-Weight policy wastes a lot of resources when the number of pending requests in a network is low for dense Mesh networks. The increase in the number of nodes also leads to an increase in wasted resources which leads to the increase in queue size. For example, when there was twenty nodes in a fully connected network there were 19 resources wasted in the first time slot but if there are now fifty nodes there could potentially be 49 wasted resources. However the increase in queue size did not lead to any instabilities. In fact, as the number of nodes increases, although the queue size increases, the queues actually become more stable which was also the case when there was only one type of resource that nodes exchange.

3.4 Ring and Star Topologies

So far most of the networks used have been generated using GRG which results in network graphs that are similar to Mesh networks as already mentioned. The Max-Weight policy was also tested in Ring and Star network topologies. The Ring topology performed better than any of the other networks tested in terms of queue size. The queues were also stable. The Star topology had a lower queue size than any of the Mesh networks tested but not as low as the Ring topology. The queues for the Star topology were also not stable.

In the Star topology nodes can only be served either by themselves or by the centre node. If the network size is large then the probability of a node on the outside being served by the centre node becomes quite low. If there are a few nodes on the outside that are not sustainable on their own then this will eventually lead to an unstable network. For the Ring topology, each node can be served by itself, one node to its left and one node to its right. This increases the chances of more nodes being served which helps the nodes with the highest number of requests to receive more resources from other nodes which improves network stability.

On the following page, In Figure 3.3 the average number of pending requests for a fully connected Mesh network and in Figure 3.4 a Ring topology can be seen. Both topologies had twenty nodes. The Ring topology results show that it is possible to have a

network that has a low number of average pending requests but also stable queues. The decrease in queue size when using a Ring topology is due to only a small number of resources being wasted. The Ring topology wasted less resources then any of the Mesh networks on average. A Ring topology with twenty nodes wasted an average of about six resources per node after 1000 time slots. A fully connected Mesh network with the same number of nodes wasted about 24 resources per node after 1000 timeslots.

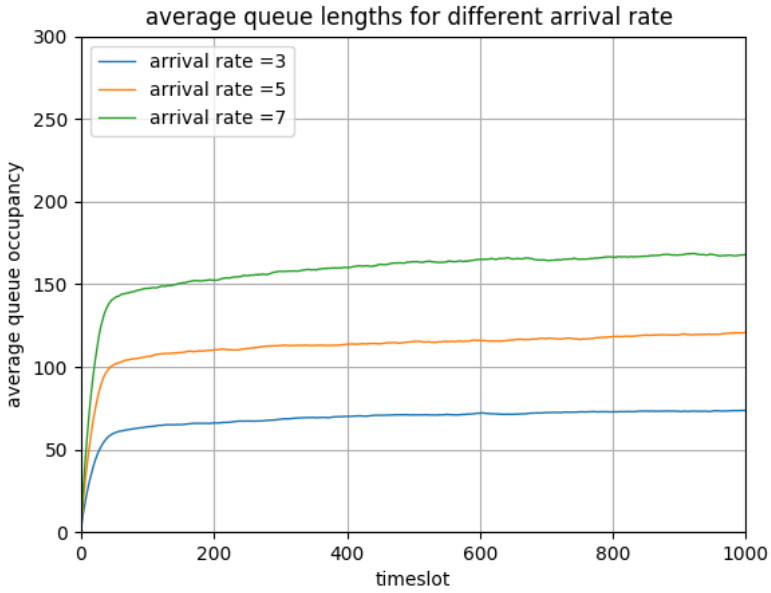

Figure 3.3 - Queues for a Fully Connected Mesh Network.

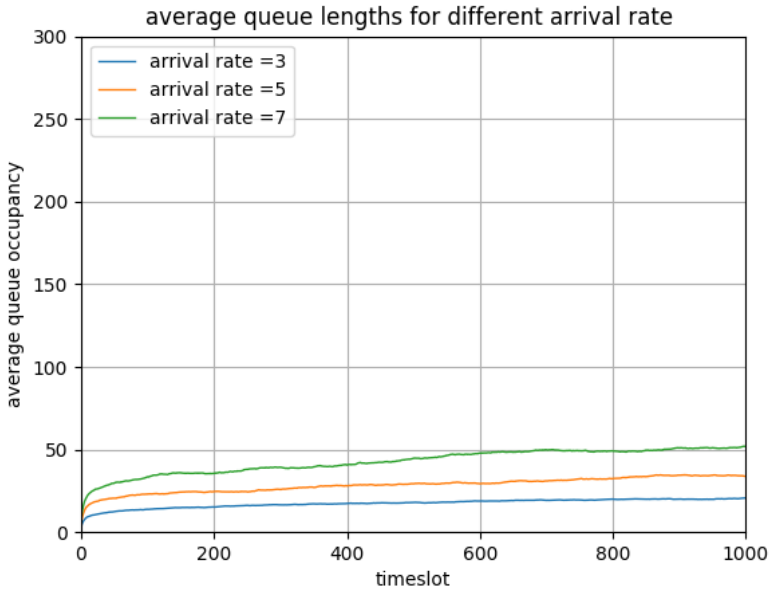

Figure 3.4 - Queues for a Ring Network.

3.5 Random Node Selection

To compare the performance of the Max-Weight policy against another node selection algorithm, it was tested against random node selection and the results of the two policies were compared. In random node selection a producer j selects a consumer i that it is connected to without any regard for that consumers pending requests or the pending requests of any other consumer in its network. At first it was thought that random node selection would cause each network tested to become unstable due to fact that producers might just keep selecting nodes with low or little pending requests. If this happens it will cause resources to be wasted. It will also mean that there is a chance that the consumers with the highest number of pending requests do not get served. If the consumers with the highest number of pending requests do not get served their queues will continue to grow. This will cause the consumers with the largest demands and their queues to become unstable which in turn will cause the network to become unstable.

However, this did not happen. On average the number of pending requests in networks that use random node selection was lower than those that use the Max-Weight policy. In fact, the average number of pending requests in the networks that use random node selection is much lower than when using the Max-Weight policy as the number of nodes N increases or both the arrival rate A_i and the production rate B_j are increased. However, the queues in the networks using random node selection are not stable. This is most likely due to the fact that in random node selection there is a possibility that resources are wasted when a producer chooses to serve a consumer with a low number of pending requests or consumers with the largest number of pending requests do not get served at all.

3.5.1 Random Node Selection Wasting Resources

When plotting the number of resources wasted in each individual timeslot by both policies, there are some spikes in the wasted resources graph for random node selection which is an indication of producers choosing to serve consumers with little or no requests. The spikes are only small and they do not occur that often but, the Max-Weight policy graphs do not contain such spikes. There are some small spikes in the Max-Weight graphs but they are towards that start of the simulations where it has already been shown in previous sections how resources are wasted by the Max-Weight policy. When plotting the accumulated wasted resources, as can be seen in Figure 3.5 on the following page, it can

be seen that random node selection wastes less resources. If we use the slope of the two curves as an indication of the rate at which resources are wasted, both policies waste resources at approximately the same rate. However, the Max-Weight policy wastes a lot more resources in the when network queue sizes are relatively as already mentioned in previous sections.

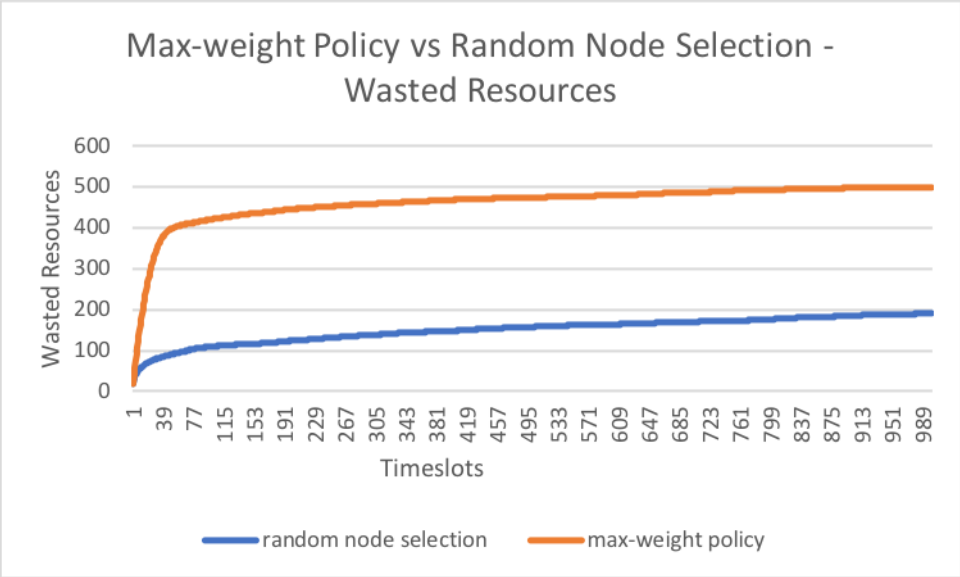

Figure 3.5 - Accumulated Wasted Resources. Max-Weight vs Random Node.

3.5.2 Random Node Selection and Ring and Star Topologies

When testing random node selection on special topologies such as Ring and Star, the Star topology performed worse than it did when using the Max-Weight policy. It also performed worse than the Mesh networks generated using GRG that used random node selection. When using a Star topology and random node selection, the queues became unstable almost straightaway. This is due to the consumers on the outside who have a high number of pending requests not receiving enough resources. It is also due to the centre node receiving more resources than it needs. The Ring topology actually performed just as good as the Mesh networks in terms of queue size and stability. However, it performed worse than a Ring topology using the Max-Weight policy in terms of stability and queue size. Again, a Ring topology shows it is possible to have a low queue size while keeping the queues stable.

3.5.3 Random Node Selection Summary

Random node selection can cause resources to be wasted if nodes with little or no pending requests are allocated more resources than they need and it can also cause networks to become unstable if nodes with high a number of pending requests do not receive enough resources. However, if does show the number of pending requests in a network can be reduced if more nodes are served in each time slot. If only nodes with a number of pending requests above some threshold are served it could reduce queue size while remaining stable.

3.6 The Average-Weight Policy

3.6.1 Introduction

In the previous sections, it was shown that the Max-Weight policy caused a lot of resources to be wasted in the first few time slots or when the queues are very low. Then it was then shown that randomly choosing a consumer to serve can decrease the number of resources wasted by increasing the number of nodes served in each time slot but it can also cause resources to be wasted in the long run. To try and increase the number of consumers served in each time slot without wasting resources and keeping the network stable, a few different approaches were taken.

The first approached involved defining a max number of producers that a consumer could be served by in each time slot. To find an appropriate value to set the max number of producers that could serve a consumer in each time slot the average vertex degree of the best performing network topologies from the Max-Weight policy simulations were first tested. The best performing network for the Max-Weight policy in terms of queue size was a Ring topology with an average vertex degree $\deg(v) = 3$. The second best performing network was a Star topology also with an average vertex degree $\deg(v) = 3$. The third best performing network was a Mesh topology generated using a GRG with radius $r = 0.25$ and average vertex degree $\deg(v) = 3$. As a result of each consumer only being able to be served be a maximum number of three producers in each time slot the average number of pending requests decreased while the queues remained stable. However, it relied on changing the Max-Weight policy to a sequential process where producers selected a consumer to serve one after the other. This takes away from the distributed nature that the Max-Weight policy offers which makes this version undesirable. It did however, highlight once again how the average number of pending requests can be reduced if more consumers are served without wasting resources and guaranteeing that the consumers with the largest demands are served.

The second approach involved choosing a consumer based on its number of pending requests and its number of neighbours. For example, two consumers might have the same number of pending requests but one consumer might be connected to a higher number of producers so the consumer with the lower number of neighbours should be selected as it has a lower probability of being served. The third approach involved choosing a consumer based on its number of pending requests and based on the number of producers that served that consumer in the previous time slot. For example, two

consumers might have the same number of pending requests but one consumer might have been served by a higher number of producers in the previous time slot and it is likely that consumer is in a good position and will receive more resources than the consumer who was served by less producers in the previous time slot.

However, both of these approaches run into the same problem as the original version of the Max-Weight policy. If we have a fully connected network, each producer sees the same set of consumers and will select the same consumer with the highest number of pending requests. So for the first approach because the network is fully connected each consumer has the same number of neighbours so that won't help a producer select which consumer to serve. In the second approach if the network is fully connected and there are two consumers with the largest demands, and one was served less than the other in the previous time slot, then each producer will select the consumer who was served less and again only one consumer will be served and there is a chance resources will be wasted.

3.6.2 The Average-Weight Policy Design

Another way to increase the number of consumers served and decrease the number of resources wasted is, if a producer finds the set of consumers that have a number of pending requests above some threshold and then randomly chooses a consumer to serve from that set. For example, if a producer calculates the average number of pending requests for every consumer in the network and then selects any consumer with a number of pending requests greater than the average that it is connected to. The way in which a producer chooses a consumer to serve is the only change that needs to be made from the initial algorithm presented in The State of Art chapter. The way in which each consumer updates its pending request and informs its neighbours about its demands at the end of a timeslot remains the same.

3.6.3 Implementation

First, a producer j finds the set of consumers N_j that it is connected to, $C = \{i \mid network.connected(i,j)\ \forall\ i \in N_j\}$. Then for each resource type $k \in K$, a producer j calculates the average number of pending requests $average_{i \in N_j} X_{ik}$ in the network. Then a producer j uses this as a threshold to find the set of consumers that have a greater number of pending requests than the threshold, $I = \{x \mid X_{xk} \geq\ average_{i \in N_j} X_{ik}\}$. Then a producer j finds the intersection of its connected consumers C and the consumers with

the highest number of pending requests I. Finally, a producer j randomly chooses a consumer i from the intersection of the two sets, $i = random(C \cap I)$.

```
1.  def get_max_pending_requests(j):
2.    # store ID of neediest consumers, one for each resource type k
3.    neediest = [−1, −1]
4.    connected = {i | network.connected(i, j)}
5.    for k = 1 : K
6.      # find all consumers with pending requests >= average
7.      average = average(X_ik(t))
8.      largest = {i | X_ik(t) ≥ average}
9.      consumers = connected ∩ largest
10.     # if producer j is connected to a consumer with
11.     # pending request >= average, randomly choose one
12.     # else service own requests
13.     if(consumers):
14.       neediest[k] = random(consumers)
15.     else:
16.       neediest[k] = j
17.   return neediest
```

Figure 3.6 - The Average-Weight Policy.

If either of the sets are empty, (the connected consumers or the consumers with a number of pending requests greater than the threshold), for any producer j, that producer will just service its own pending requests. This only occurs in networks where there are nodes with very few or no neighbours.

This approach can be built on further if it is assumed the number of pending requests follows a normal distribution. When plotting a histogram of the number of pending requests in a network that is using the Max-Weight policy it can be seen that the number of pending requests does closely follow a normal distribution. If each producer j also calculates the standard deviation σ of the pending requests X_{ik} and then only chooses the set of consumers that have a number of pending requests $\geq average_{i \in N_j} X_{ik} + 2\sigma$ where $average_{i \in N_j} X_{ik} + 2\sigma$ is the new threshold, it will only select consumers with the highest number of pending requests in the network. But if a producer uses this threshold then less consumers will be served in each time slot and resources will be wasted. If the threshold is changed to $threshold = X_{xk} \geq average_{i \in N_j} X_{ik} + \sigma$ then more consumers will be served and less resources will be wasted. In fact if the threshold is changed to $X_{xk} \geq average_{i \in N_j} X_{ik} - \sigma$ or even $X_{xk} \geq average_{i \in N_j} X_{ik} - 2\sigma$ then even more consumers will

be served and less resources wasted. If the $threshold = X_{ik} \geq average_{i \in N_j} X_{ik} - 3\sigma$ then the policy is essentially the same as random node selection.

There is basically a trade-off between wasting less resources and guaranteeing that the consumers with the highest requests are served in each time slot. Choosing the average number of pending as the threshold wasted the least amount of resources while still ensuring stability instead of calculating the standard deviation.

3.6.4 Evaluation

To evaluate the Average-Weight policy, it was tested exactly how the Max-Weight policy was tested in this dissertation. The Average-Weight policy was tested in a Ring, a Star and a range of Mesh topologies. Similar to the Max-Weight policy, as the Mesh networks moved from very sparse to fully connected the networks became more stable. However, the queues for the networks using the Average-Weight policy were much lower than when using the Max-Weight policy as the Mesh networks became more dense. For the sparse Mesh networks both policies performed similar both in terms of stability and queue size. The Average-Weight policy cut the queue sizes by more than half as can be seen if we compare Figure 3.3 on page 26 with Figure 3.7 on the following page for a fully connected Mesh network. Most importantly the queues were still stable.

The Max-Weight and Average-Weight policy performed similar in terms of both queue size and stability for a Ring topology. However, the Max-Weight policy performed better in terms of queue size than the Average-Weight policy for a Star topology. The Star topologies tested using the Average-Weight policy also became very unstable a lot quicker than the Star topologies that used the Max-Weight policy.

3.6.5 Conclusions

The Average-Weight policy allows for more consumers to be served by producers in each time slot compared to the Max-Weight policy in the dense Mesh networks. Instead of selecting the consumer with the largest pending requests, it selects a consumer with a number of pending requests greater than or equal to the average number of pending requests in a network. As more consumers are served in each time slot there is lower probability of resources being wasted than when only one consumer is served. As less resources are wasted more requests can be serviced in each time slot. As more requests are serviced the queues sizes decrease. As the queue sizes decrease requests spend less

time in the network waiting to be service which could potentially speed up the time taken to execute a task.

The Average-Weight policy can be implemented as a distributed algorithm which is one of the reasons the Max-Weight policy was chosen in the first place. The Average-Weight policy does not require any information about a nodes production and generation rates when selecting who should be served in each time slot just like the Max-Weight policy. The Average-Weight policy has also been able to stabilise every network that the Max-Weight policy has also been able to stabilise which is probably the most important point.

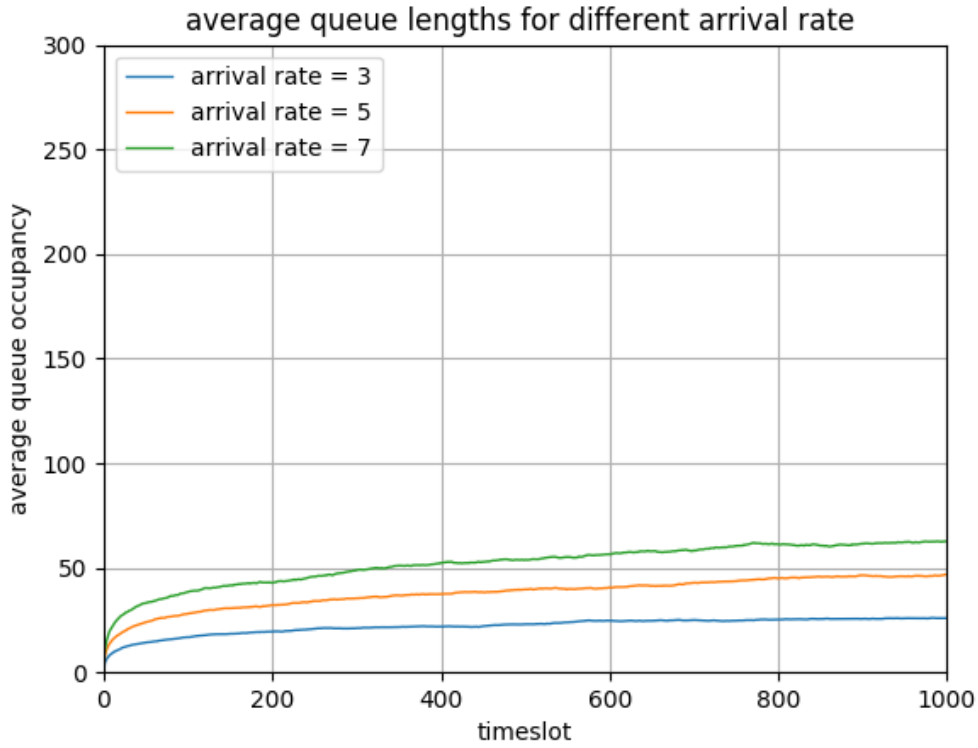

Figure 3.7 - Average Weight Policy on a Fully Connected Mesh Network.

4 Production Planning

4.1 Introduction

When the number of resource types K that nodes exchange among one another increases to more than one, each producer j is allowed to do some planning to decide how to generate resources to best serve the neediest consumers. Each producer j has a set of different possible production plans P_j it can choose from during each time slot. Under each production plan $p \in P_j$, a producer j produces B_{jk}^{p} units of resource $k \in K$. A producer can increase the production of one resource type by putting more time and effort into the production that resource, at the expense of decreasing the production of other resource types. [8]. One way to think of production planning is to imagine that a producer has ten units of energy. The producer then has to decide how to use this energy to execute different tasks. For example, if we consider a sensor that is used to measure air quality in an urban environment, the sensor can decide how much energy it should use to either record measurements for itself or to help other nodes by exchanging measurements with them. In this chapter, the benefits of producers scheduling the production of their resources will be presented. The drawbacks of producers choosing not to use production scheduling will also be presented as a comparison.

4.2 Production Rates

In this dissertation, the production rate and arrival rate of a network have assumed to be within the same range. As in if the max arrival rate of a network is $A_{max} = 10$ and consumers can generate $0 - 10$ requests for a resource then the max production rate of that network should also be $B_{max} = 10$ so that producers can also generate $0 - 10$ resources to service those requests. Of course, different producers and consumers can have different arrival rates and production rates in different time slots due to different constraints. Also, different devices can generate different types of resources. For example, a high-performance computer should be able to execute more tasks in the same amount of time as a smart phone.

If the arrival rate of the network is greater than the production rate of that network, then the average number of resource requests will be greater than the average number of resources generated in the entire network. This will soon cause the network to be overwhelmed with more requests then it can handle and the network will eventually

become unstable. If the arrival rate is less than the production rate, then the average number of resource requests will be less than the average number of resources generated in the entire network. Which is of course good because it will lead to a sustainable environment but it is probably not a realistic assumption for most networks. The effect of having a higher, a lower and an arrival rate equal to the production rate can be seen in Figure 4.1 on the following page. The production rate of the networks used in these simulations was set to $B_j \leq 2$, the number of nodes $N = 20$ and number of resource types $K = 2$.

When the arrival rate is greater than the production rate, the network becomes unstable almost straight away. This is because the number of resources generated is greater than the number of resource requests 0.0% of the time on average. Which if we consider the conditions for sustainability (Equation 4.1), which states that the conditions for sustainability can be shown if the total number of resource requests is less than or equal to the total number of resources generated [8], then the network cannot be stable when the arrival rate is greater than the production rate.

$$\sum_{i \in N_j} a_i \leq \sum_{j \in N_i} b_j$$

Equation 4.1 - The Conditions for Sustainability

When the production rate is greater than the arrival rate, it appears as though the queues will never grow to infinity. This is because the number of resources generated is greater than the number of resource requests 99.9% of the time. The network can become unstable even if the number of resources generated is greater than the number of resource requests in certain situations. One way it can happen is if the neediest nodes in a network do not receive enough resources, due to different network topologies or a poor node selection algorithm, and their queues continuously grow.

When the arrival rate is equal to the production rate, the number of resources generated is actually greater than the number of resource requests just over 50% of the time on average. The big spike in the first few time slots is due to the Max-Weight policy wasting resources which was explained in Chapter 3. After the first few time slots the queues start to become more and more stable. Each different scenario was simulated on fifty different networks to get an average which causes the graphs to look smooth.

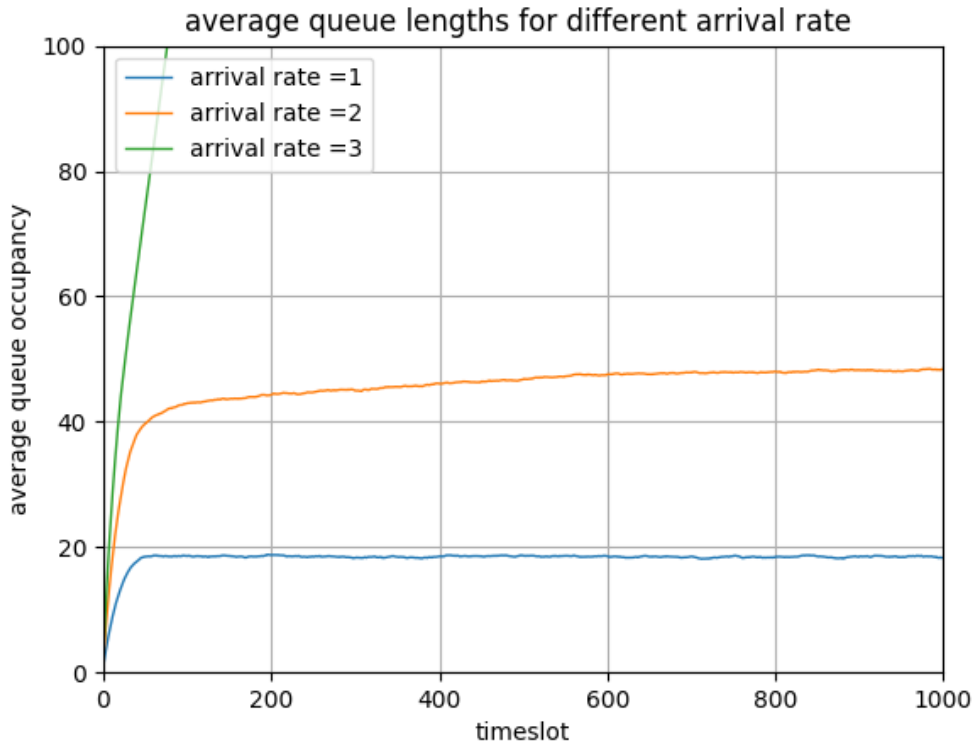

Figure 4.1 - Arrival Rate vs Production Rate.

4.3 Production Plans

The number of plans that a producer can generate is related to its production rate. In other words, when the production rate increases, each producer should be able to generate more plans. Each plan represents a different way that a producer can use the resources it has to serve the consumers with the largest demands. For example, imagine in a smart home a solar panel has converted the sun's energy into renewable energy. Imagine that the solar panel is a producer with a production rate $B_j \leq 2$ and it produces two units of renewable energy. These two units can be converted into either two units of electricity (solar photovoltaics) or two units of hot water (solar thermal) or one unit of each [11]. However, if the production rate of the producer was $B_j \leq 1$, the producer can only generate one unit of either resource type. As the production rate increases so does the number of plans that a producer can generate. In Table 4.1 on the following page, different ways that a producer could split their resources based on their production rate can be seen. As the number of resource types increases the number of plans also increases.

Production Rate	Number of Plans	Plans
1	2	[0,1], [1,0]
2	3	[0,2], [1,1], [2,0]
3	4	[0,3], [1,2], [2,1], [3,0]

Table 4.1 - Production Rate vs Production Plans

4.4 Impact of Plans

To simulate the production capabilities of different producers at various times, the total number of production plans that a producer could generate was varied for producers between time slots. In other words, to represent producers not being able to produce certain resource types at different times, due to other commitments outside of the exchange scheme or due to device constraints, the number of plans a producer could generate was different for a producer in different time slots.

Each producer was able to generate between one and some maximum number of plans. The maximum number of plans depended on the production rate of that producer. If there are producers in a network that do not generate any plans at all it can cause the network to become unstable very quickly but this will be discussed more later. When a producer is able to generate more plans, the producer increases the chances of generating a plan that will best serve the consumers it has chosen to serve increases.

4.5 Scheduled Production Planning

When a producer generates more than one plan it is able to do some planning to select the plan that will best serve the consumers it has chosen to serve. To select the plan that will best serve the consumers that a producer has chosen to serve, a producer simply loops through each of the plans it can generate and chooses the plan that will cause the most requests to be served.

To calculate which plan will cause the most requests to be served, a producer starts with the first plan it can produce. The producer then multiplies the value of each resource type in that plan by the value of the corresponding resource type of the consumer it has chosen to serve. Next the producer sums up all the totals for each resource type to calculate the total value for that plan. Then it repeats this process for each plan it can generate. Finally, it chooses the plan that has the highest total and uses that plan to serve the consumers with the highest requests. Pseudocode of this function can be found in

Figure 1.2 below. A walk through of how the function is implemented can be found in Table 4.2 below. Plans are the set of production plans a producer can generate. Requests are the requests of the consumers that a producer has chosen to serve.

```
1.  def select_production_plan(production_plans,neediest_consumers):
2.  # select most rewarding plan
3.  plan_totals = [0 for p = 1 : P]
4.  for p = 1 : P
5.      total = 0
6.      for k = 1 : K
7.          total += production_plans[k] * neediest_consumers[k]
8.      plan_totals[p] = total
9.
10.     best_plan_index = max(plans_totals)
11.     plan = production_plans.index(best_plan_index)
12.
13. return plan
```

Figure 4.2 - Selecting a Production Plan - Function.

Plans	[2,0]	[1,1]	[0,2]
Requests	[3,1]		
Subtotals	[2*3,0*1] = [6,0]	[1*3,1*1] = [3,1]	[0*3,2*1] = [0,2]
Total	6	4	2

Table 4.2 - Selecting a Production Plan – Walkthrough

The *select_production_plan* function, takes the production plans that a producer has generated and a list of tuples containing the consumers it has chosen to serve. Each tuple contains the index of the consumer in the network it has chosen to serve and the value of its highest request. In the first position of *neediest_consumers* list the consumer with the highest requests for resource type $k = 1$ is stored. In the second position the consumer with the highest requests for resource type $k = 2$ is stored and so on up to resource type $k = K$. On line three, *plan_totals* are a list used to store a total for each plan. The total for all plans is calculated lines 4-9. The total for each individual plan is calculated lines 6-7. The plan with the highest total is the plan that will best serve the consumers a producer has chosen to serve.

4.6 Selfish Production

So far it has been assumed that each producer will cooperate fairly and try its best to produce plans that will be of benefit to the consumers it has chosen to serve. However, if a producer is to act selfishly and only produce plans that benefit themselves in some way it can cause the network to become unstable over time. In this dissertation, there is no way to incentivise producers to cooperate fairly by inflicting some sort of cost on producers who do not cooperate. Instead the damage that a producer can cause to its network by acting selfishly is presented.

In selfish production planning each producer only generates plans that only represent their own needs and uses these to serve the consumers it has agreed to serve. To simulate nodes that act selfishly, two different production scheduling algorithms were used. The first is the production scheduling algorithm described in the previous section. The second is a selfish production algorithm where producers select plans based on their own requests. The selfish production function is identical to the one described in the previous section expect that a producer passes the production plans it has generated and its own pending requests to choose a plan. Both the Max-Weight policy and Average-Weight policy were tested. First the percentage of nodes in each network that choose the selfish production algorithm in each time slot was set to 25% and then increased to 50%, then 75% and then finally 100%. As the percentage of selfish nodes in the network increased, the networks went from stable to unstable. However, the queue sizes when using selfish production scheduling were also slightly lower than the queue sizes when using scheduled production, which was not expected.

The decrease in queue sizes are due to the fact that selfish production can cause less resources to be wasted and as previously shown wasting resources can lead to an increase in queue size. For example, when using the Max-Weight policy in a full connected Mesh network each producer selects the same consumer to serve as previously shown. If each producer uses production scheduling to serve a consumer, each plan generated by every different producer will be similar to each other in an attempt to serve the demands of the consumer it has selected as best as it can. For example, if each producer selects consumer i and consumer i has 10 requests for resource type $k = 1$, each producer will try and generate a plan that can service as many of those 10 requests as possible. If there are 10 nodes in the network and they are able to generate 10 resources for resource type $k = 1$, then 90 resources will be wasted. However, if there are a few selfish nodes they may generate resources for resource type $k = 2$, but still choose to serve consumer i then

less resources maybe wasted if consumer i has requests for resource type $k = 2$ aswell. This only really occurs in the first couple of time slots, as can be seen in Figure 4.3 on page 43. After this selfish production, actual wastes resources at a higher rate.

Even though a producer might be using selfish production, due to the fact that the Max-Weight policy and the Average-Weight policy chooses the consumers with the highest requests, these consumers will still get served which causes the networks to remain stable. Also, sometimes a consumer that a producer has chosen to serve might have similar needs to itself. This means that a consumer might receive resources similar to its needs. At times, a producer using selfish production will choose a plan that doesn't reflect the needs of the consumer it has chosen to serve which causes resources to be wasted which can lead to instabilities. However, in general as long as the nodes with the highest requests are chosen and receive some resources for the correct resource type that it is in need of, it will stop them from becoming unstable. But in the long run, nodes choosing selfish plans in the network can cause the network to become unstable.

4.7 Random Production

Another way that producers can act selfishly and can cause the network to become unstable is by choosing random production. In random production producers just choose any plan without any concern for the needs of the consumers with the highest requests or themselves. One way of thinking of random production is that producers choose a plan that would require minimal cost for them to produce. As with the selfish production, two production algorithms were used. The first was the scheduled production planning algorithm described previously. The second was a random production function. To simulate random production, a producer just selected a random plan from the set of plans it generated and used that to serve the consumer it has chosen to serve. Again, both the Max-Weight policy and the Average-Weight policy were tested. Again, the percentage of nodes that used random production selection was set to 25%, then 50%, then 75% and finally 100%.

For both the Max-Weight policy and Average-Weight policy, as the percentage of nodes using random production scheduling increased the networks became unstable almost straight away. For both policies when the percentage of nodes using random production scheduling was greater than 50% the networks became completely unstable almost straight away.

When using the Max-Weight policy and the Average-Weight policy there is a chance that the consumer that a producer chooses to serve is the same node. So, when using selfish production, if a producer chooses a consumer to serve that happens to be itself, it chooses a suitable production plan. However, in random production planning due to the fact that the plan chosen is not based on the needs of a consumer, the number of resources wasted is very high. Even if the nodes with the highest requests are served, there is a good chance that the plan selected will not reflect their needs most of the time. This causes the number of requests at the consumers with the highest demands to continue to grow which can lead to the whole network becoming unstable. Random production planning can cause the network to become unstable very quickly compared to selfish production and scheduled production planning.

4.8 Free-Riding Nodes

As mentioned previously, the more plans that a producer generates helps the network to become more stable. It is acceptable that in some time slots there might be some producers that do not contribute any resources. However, if there are a number of producers who do not contribute any resources and they're only a part of the network to receive resources then the network will become unstable very quickly. In this dissertation, there is no way to inflict a cost on producers who do not contribute any resources so instead the effects of producers who do not contribute any resources is presented.

To simulate free-riding nodes, a percentage of nodes in each network did not do any resource generation or try to serve any consumers. Even when the percentage of free-riding nodes in each time slot was set to as little as 5% the networks became unstable very quickly.

In Figure 4.3 on the following page, an example of scheduled production, selfish production (50% selfish nodes), random production (50% random nodes) and free-riding (2%) nodes can be seen for various arrival rates. Each scheduling algorithm was simulated on fifty different networks with twenty nodes. Each network used a fully connected Mesh topology. Each simulation used the Max-Weight policy to select which nodes to serve.

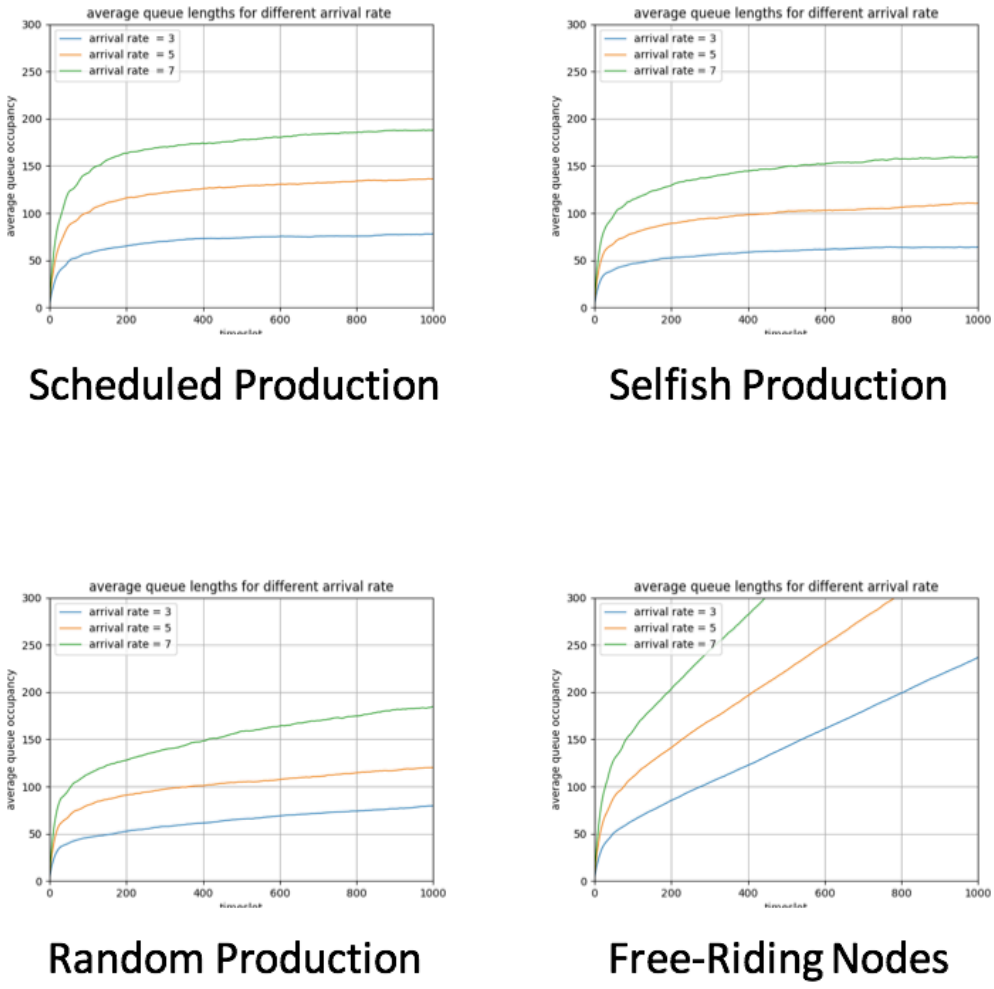

Figure 4.3 - Production Scheduling Algorithms.

4.9 Conclusions

In this chapter, the basic idea of producers scheduling the production of their resources to help achieve a sustainable network was discussed. Then a simple function to select a plan that will best serve the consumers that a producer has chosen to serve was presented. Finally, the disadvantages of not using production scheduling were shown. Selfish production did decrease the queue sizes but in the long run it could lead to a network becoming unstable. If there are nodes in a network that use random production scheduling it will lead to instabilities. Even a small number of free-riding nodes in a network will also cause the network to become unstable.

An outline of how both the Max-Weight or Average-Weight policy and production scheduling work together can be found in Figure 4.4 below. The *get_max_pending_requests()* function can be either the Max-Weight or the Average-

Weight policy. The Max-Weight pseudocode can be found in Chapter 2 and the Average-Weight pseudocode can be found in Chapter 3. More details about how

1. *# time-slotted algorithm*
2. $t \in \{0,1,2,\dots,T\}$;
3. *# each producer selects its consumers to*
4. *# serve and its production plan*
5. **for** $j = 1 : N$
6. Find $\hat{\imath} = \text{get\_max\_pending\_requests}(j)$
7. Find $p^* = select\_production\_plan(plans, \hat{\imath})$
8. **for** $i^* = i \; in \; \hat{\imath}$
9. Set $I_{ji^*}(t) = 1$
10.
11. *# each consumer informs its neighbours about its demands*
12. **for** $i = 1 : N$
13. $X_i(t) = \left[\max\left(0, X_i(t-1) - M_i(t)\right)\right] + A_i(t)$
14. Send $X_i(t)$ to every producer

Figure 4.4 - Node Selection and Production Scheduling

5 Graph Properties

5.1 Introduction

In this dissertation a number of graph properties and graph characteristics were studied to see if they had an effect on network stability. The main property studied was assortativity as described below. Some other properties studied included density. Density is closely related to the impact that increasing the radius of the GRG, used to simulate Mesh networks, had as was shown in Chapter 3. When the radius of the GRG increases, the number of edges increases and the density also increases. In Chapter 3 it was shown that an increase in edges helped increase stability, which implies that networks that have a high density are more stable than networks with low density. However, a Ring topology has a relatively low density score but it is also stable, so low density does not necessarily lead to any instabilities.

5.2 Assortative Mixing

Assortative mixing is a term used to describe selective linking among nodes according to a certain characteristic [16]. Such node characteristics include the degree of a node, where degree indicates the number of edges connected to a node in an undirected network. In a directed network a node has an in-degree and out-degree characteristics. If nodes in a network of the same degree tend to be connected to one another then the network is said to be assortative. For example, if nodes of high degree connect with other nodes of high degree rather than connecting with nodes of low degree. However, if nodes of low degree tend to connect with nodes of high degree and nodes of high degree connect with nodes of low degree then the network is disassortative.

The iGraph package contains functions to calculate the assortativity coefficient of a network. The assortativity coefficient measures the correlation between node characteristics in a network. The iGraph function $assortativity()$ uses the Pearson correlation coefficient to calculate the assortative coefficient in undirected graphs, which have mainly been used. The $assortativity()$ function returns a value between -1 and 1 known as the r score. When $r = 0$, then the network is neither assortative nor disassortative. Positive values of r indicate assortative mixing whereas negative values of r indicate disassortative mixing [17].

The $assortativity()$ function can measure the assortativity coefficient for any node characteristic. In this dissertation, node degree is used to see if assortative or disassortative networks perform better for the Max-Weight policy. To calculate the assortativity coefficient for the node degree of a network, the node degree of every node in the network is passed as an argument to $assortativity()$ function.

The GRG, or Mesh networks, that have been used in the simulations in this dissertation so for, all have r scores between 0-1. The sparse Mesh networks actually have an r score closer to 1 than the ones that have a high number of edges between nodes. As the average node degree of a Mesh networks gets higher, the r score gets closer to 0. A fully connected Mesh network does not have an r score as every node has the same degree. For the Mesh networks in this dissertation, r scores close to 1 have actually lead to unstable networks. For a Ring topology, it also does not actually have an r score as every node has the same degree. A Star topology is an example of a disassortative network with perfect disassortative mixing and with r score equal to -1.

To further test assortative mixing and whether it correlated with stability, two very similar networks in terms of size, density and degree distribution but very dissimilar assortative r scores were tested. Both networks had fifteen nodes and a density value of 0.171. The assortative network had an r score equal to 0.171 and the disassortative network had an r score equal to -0.744. Both networks were tested using the Average-Weight policy. Both networks performed similar in terms of queue size and stability. Both networks had unstable queues as can be seen in Figure 5.1 on the following page along with their respective topologies. Both networks only contained fifteen nodes and in previous sections it was shown that an increase in the number of nodes helped to stabilise the queues so the low number of nodes in these two networks could be the reason for them being slightly unstable and not their r scores. Also, the assortative network wastes slightly more resources on average per node. In the graphs of the two topologies, a thick edge indicates a consumer being served by a producer, the vertex label and colour indicates the number of pending requests. The darker the shade of blue means more requests.

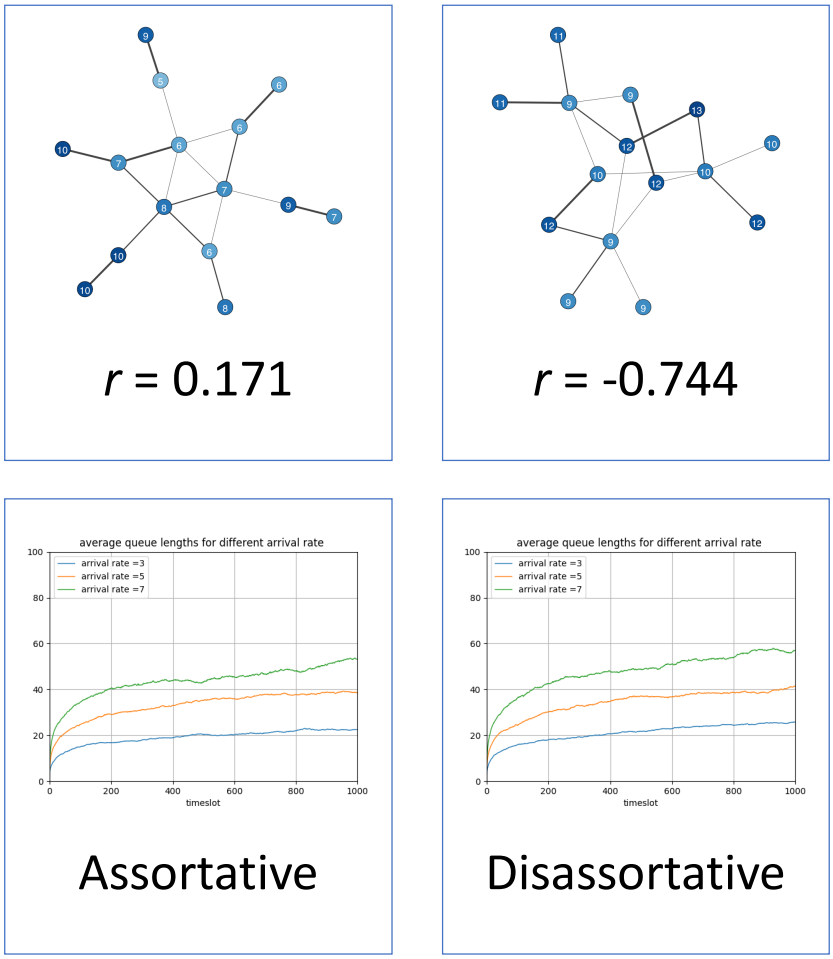

Figure 5.1 - Assortative vs Disassortative Networks.

The r score of an assortative network (top left) and a disassortative network (top right). The average number of pending requests of the assortative network (bottom-left) and the disassortative network (bottom right).

As already mentioned, most of the Mesh networks used in the simulations in this dissertation have had an r score between 0-1 making them assortative networks as opposed to disassortative. Generating disassortative networks was actually quite difficult and required edges to be hardcoded most of the time. A Star topology is one way of generating disassortative networks. But as already seen in previous sections Star topologies do not always lead to stable networks. To improve on a Star topology a small number of high degree nodes can be placed at the centre of a large number of low degree nodes. The high degree nodes in the middle would be connected to every low degree node on the outside. The low degree nodes on the outside would only be connected to the few nodes at the centre. Generating a network this way results in a perfect disassortative score ($r\ score = -1.0$). The number of high degree nodes in the middle also had an effect on stability. If just under half of the nodes in the network where placed in the centre to serve the low degree nodes on the outside, the network became stable. Placing a few nodes in

the centre, as can be seen in Figure 5.2 below, as opposed to just one, like a Star topology, resulted in a stable network.

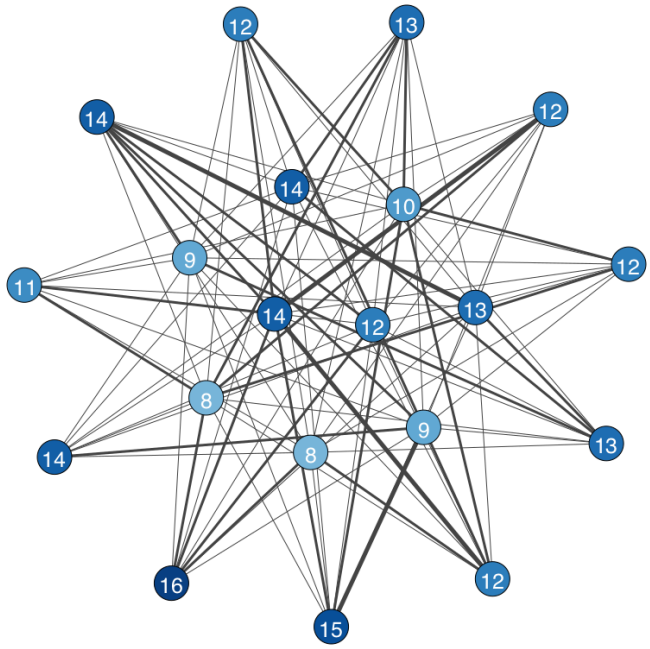

Figure 5.2 - Perfect Disassortative Mixing.

5.3 Conclusions

In general, the assortative and disassortative networks performed very similar, which implies that there may not be a link between assortative mixing and stability or disassortative mixing and stability. But this is only based on the networks mentioned above. However, the best performing networks in terms of stability seen so far have been a fully connected Mesh network and a Ring network. Both of these networks do not have an r score because every node in these networks have the same degree. This could imply that networks that do not display either assortative nor disassortative mixing are the most stable. The best performing assortative or disassortative network in terms of stability was the Star like disassortative network with a number of high degree nodes in the centre serving the requests of low degree nodes on the outside. If there was a way to strategically place the most resource nodes in the centre to serve the nodes that generate the most requests on the outside, that could lead to sustainable environment.

6 Example Applications

6.1 Introduction

In this chapter, two hypothetical examples of applications using the Max-Weight policy and production scheduling are provided. The two applications are based on two different networks generated using from two different datasets. The two datasets are basically just two files (one GraphML file and one text file), with a collection of vertices and edges. A network was then generated for each of the datasets. The Max-Weight policy was tested on both networks. The datasets did not provide any information about the arrival rate or production rate of the networks. Instead the arrival rates and production rates were generated using random number generators just like in every other simulation seen throughout this dissertation.

6.2 Power Grid

The first dataset is an undirected network representing the topology of the Western States Power Grid of the United States [18]. Which is the electrical power grid for the western states in the United States. Each vertex is either a generator, a transformer or a substation and each edge represents a power supply line. This dataset was chosen as a number of examples used to describe different aspects of this dissertation have revolved around smart cities, smart grids and renewable energy. The full dataset contains over 4900 vertices, but only a number of small subgraphs were tested.

As an example, imagine that the network is a smart grid and each vertex represents a smart home. The resource that the smart homes exchange between one another is excess electricity generated from renewable sources. Each house exchanges its excess electricity to their neighbours that they are connected to so that electricity is not wasted or that houses that have high demands have enough supply. In Figure 6.1 on the following page, it can been seen that the disconnected nodes in the network have the highest demands for electricity. The number displayed on each vertex and their colour represents their demands, i.e. a darker shade of blue represents a higher number of demands. A thick edge represents an exchange of resources. Although, some of the disconnected nodes have a low number of requests, the disconnected nodes with a high number of demands are being starved of resources. This will cause the network to become unstable over time. The disconnected nodes would cause the network to become unstable

regardless of the node selection policy used, i.e. the Max-Weight policy or the Average-Weight policy. The nodes who are connected to even a small number of other nodes have a much lower number of requests. In Figure 6.2 on the following page, a much larger subgraph is generated with around 280 nodes. This graph does not contain any disconnected nodes and its queues were more stable. However, there are a number of nodes with only a one or two neighbours and some of these nodes have a high number of requests. This would eventually cause the network to become unstable over time. Whereas the majority of nodes with a few neighbours have a low number of requests which implies that there individual queues are stable.

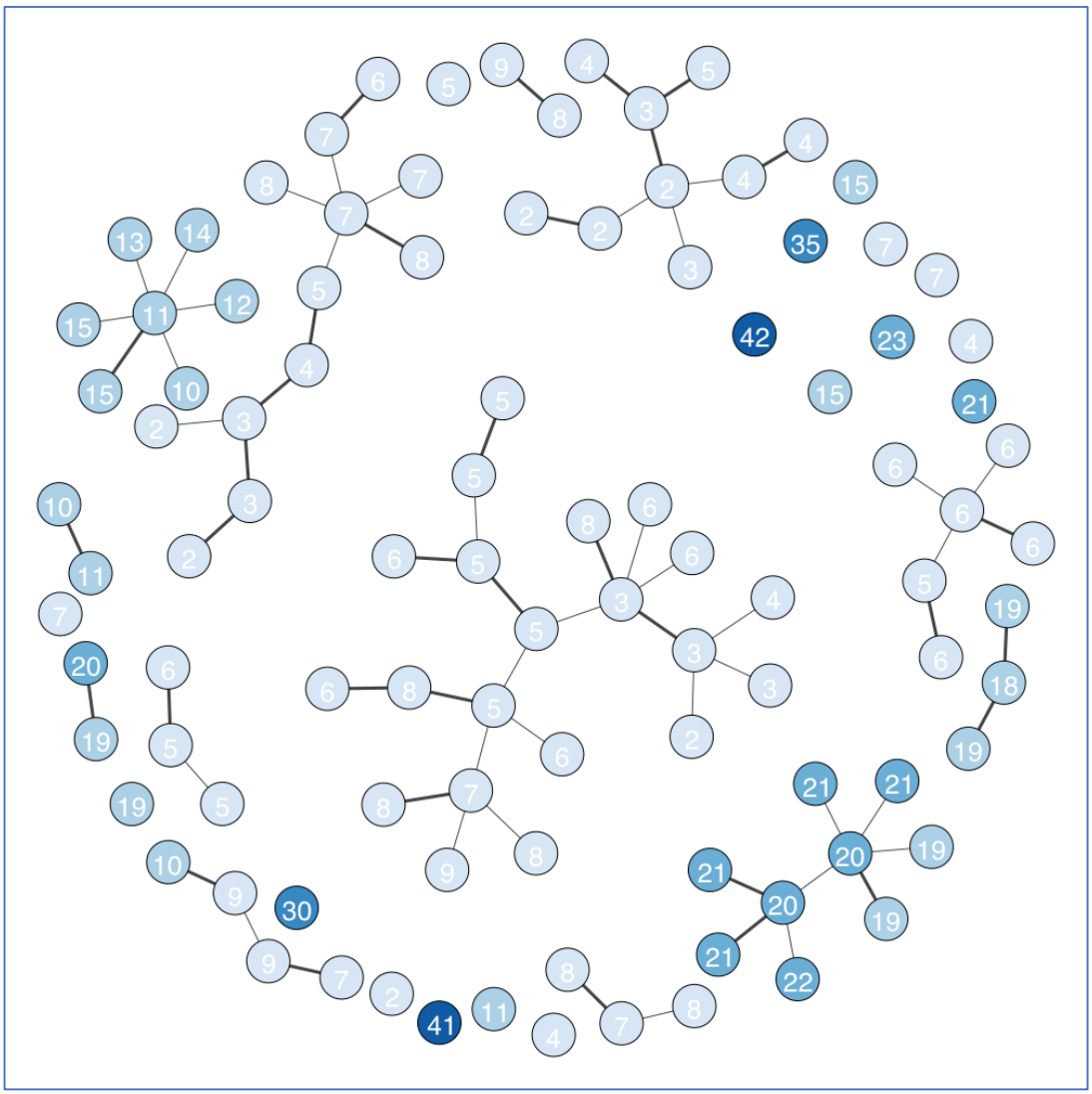

Figure 6.1 - Western States Power Grid of the United States. Subgraph 1.

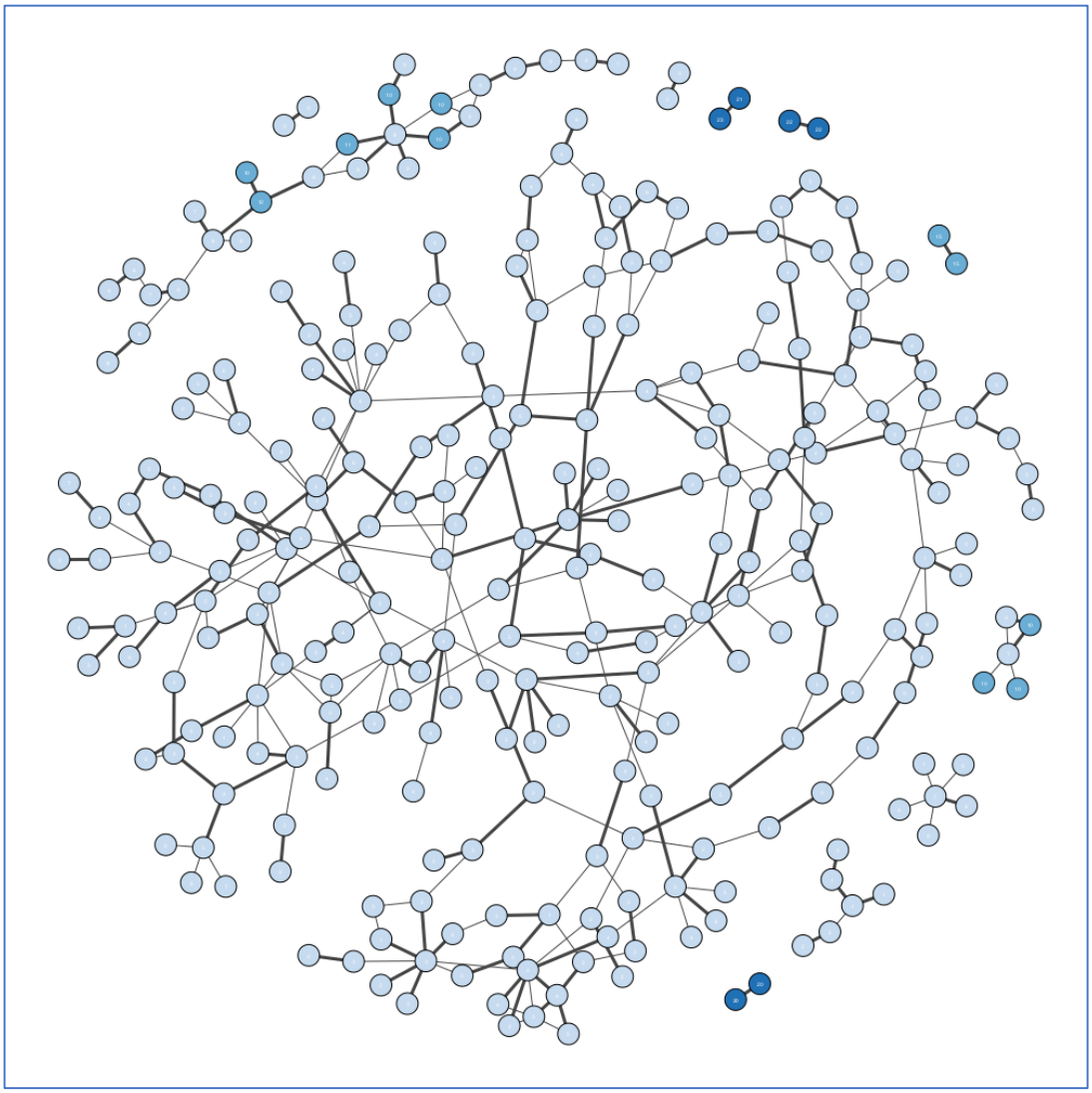

Figure 6.2 - Western States Power Grid of the United States. Subgraph 2.

6.3 Favour Exchanging

The second dataset is an undirected network containing an anonymized list of people who are friends on Facebook [2]. It is essential a list of numbers and edges. Each vertex represents a Facebook profile and each edge represents two profiles who are friends with each other. The full dataset contains over 4000 vertices but only a small number of vertices were used to generate some small subgraphs. In Figure 6.3 on the following page, the nodes with the highest number of requests are the nodes with a relatively low number of neighbours. Although there are some nodes with a low number of neighbours and a low number of requests, there are no nodes with a high number of neighbours and a high number of requests. In this application, members of the social community could exchange different favours with another. For example, in a network of college students, students

exchange lessons with one another in different subjects measured by some unit of time. The disconnected nodes with little or no requests, in the graph below, would be smartest students in their course who do not need any help. Whereas the nodes with the highest number of requests are the students who are struggling most and who need help. The students with the highest requests also have a low number of neighbours. In this example, the number of requests could be hours of study needed and different resource types could be different subjects.

Again, the disconnected nodes and the nodes with low degree would cause the network to become unstable. Both the Max-Weight policy and the Average-Weight policy would perform similar due to the disconnected and low degree nodes causing the queues to become unstable.

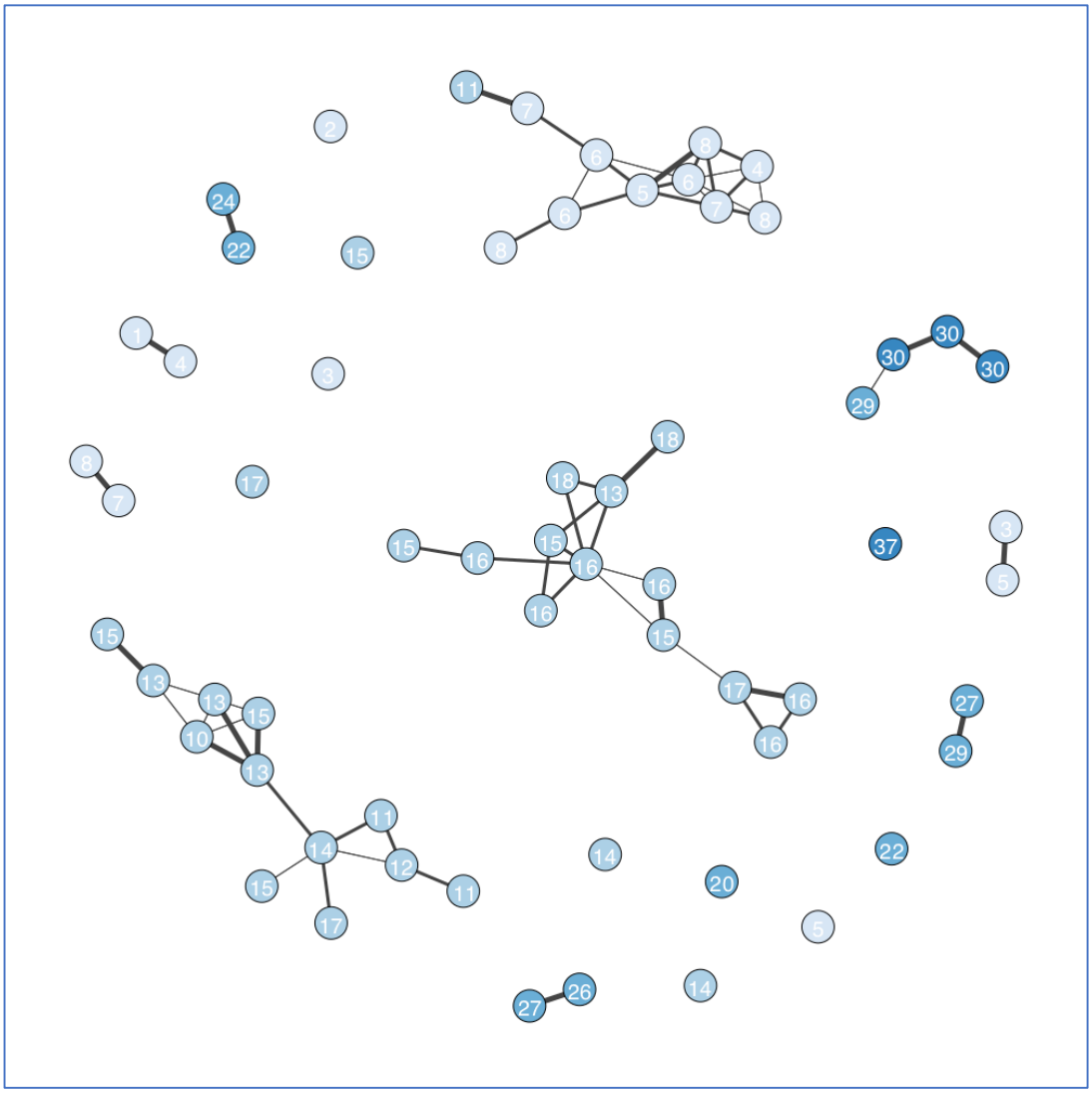

Figure 6.3 - Social Network Graph.

7 Conclusions

7.1 Findings

In this dissertation the Max-Weight policy was presented as a way for IoT devices to cooperate and exchange resources. The Max-Weight policy was analysed in different network topologies. We found that the best performing network topologies in terms of stability for the Max-Weight policy to be deployed in are a fully connected Mesh topology and a Ring topology. However, we found that a network arranged in a Ring topology has a much lower queue size compared to a network that is organized in a Mesh topology. For example, a fully connected Mesh network has a queue size almost twice as big as a network with a Ring topology structure. The Mesh network queues were much bigger due to the fact that the Max-Weight policy causes resources to be wasted as Mesh networks become more dense. As the number of edges increase in a Mesh network less and less nodes are served. For example, in a fully connected Mesh network, each producer sees the same set of consumers which means each producer will serve the same consumer with the highest demands. This means that only one consumer will be served in each time slot which can lead to resources being wasted if the aggregated total of resources a consumer receives is greater than its number of demands . Wasting resources leads to an increase in queue size. The Max-Weight policy did cause resources to be wasted in a Ring topology network structure but far less compared to most Mesh networks.

As a solution to avoiding wasting resources, the Average-Weight policy was introduced. The Average-Weight policy reduced the amount of resources wasted by increasing the number of consumers served in each time slot. Reducing the number of resources wasted in each time slot also led to a decrease in the average number of pending requests at the end of each time slot. It is important to note that a decrease in queue size does not necessarily lead to a more stable network though, as was shown by random node selection which had the lowest of queue sizes but it did not lead to stability. As the Average-Weight policy leads to much lower queue sizes, it means that requests spend less time waiting to be serviced and as a result more jobs could potentially be executed in roughly the same time as a network that deploys the Max-Weight policy. However, the Max-Weight policy performed similar to the Average-Weight policy in a network with a Ring topology in terms of both stability and queue size. A Ring topology already ensures that each producer cannot serve the same consumer due to its structure.

The Average-Weight policy actual caused a network with a Star topology to become very unstable. However, the Max-Weight policy did not stabilise networks with a Star topology either so maybe Star topologies should be avoid when we want to guarantee stability.

The Max-Weight was chosen to tackle the resource sharing problem in IoT as it had three properties that make it advantageous. It is distributed, it does not require a priori knowledge about nodes production and demand generation rates and it is throughput optimal in that it stabilises a system whenever possible. The Average-Weight policy is also distributed and does not require a prior knowledge about nodes and so far it has been able to stabilise every system that the Max-Weight policy stabilises.

We then built on top of our node selection algorithms and extended to the case where there is more than one type of resource that nodes exchange. Each producer is now allowed to do some planning to decide how best to uses it resources to best serve the consumers with the largest requests. Although production scheduling may increase a nodes work load it was shown that production scheduling is needed to ensure stability. Whereas when there are a number of nodes in a network who act selfishly, either by only producing plans that reflect their own needs or are low cost for them to produce it can also cause their network to become unstable very quickly.

We also studied assortative and disassortative mixing based on node degree and the impact it has on network stability. Some dissassortative networks performed very well in terms of stability. Whereas most networks that had an assortative score were unstable. However, a Ring topology and a fully connected Mesh network are examples of networks that are neither assortative nor disassortative but are stable which implies assortative mixing may not be linked with stability.

Finally, both the Max-Weight policy and production scheduling were applied to two hypothetically applications using two different subgraphs of two actual network topologies. Network stability was not achievable in either network due to disconnected nodes and nodes with a low number of neighbours who were not sustainability on their own. Whereas, in both networks, the nodes with a high number of neighbours were able to use the Max-Weight policy to exchange resources and ensure stability for themselves.

7.2 Future Work

As this dissertation is mostly theoretically based, the next steps would be to apply that theory to some actual IoT network data. Although in Chapter 6, both the Max-Weight policy and production scheduling were applied to actual network graphs, the arrival rates

of each consumer and the production rates of each producer were both simulated using random number generators. Ideally, we would like to learn more about the production rate and the arrival rate of different IoT networks and the devices used in those networks and then apply the Max-Weight policy to see if the results presented in this dissertation will hold.

Further investigations into what length of time one time slot is equivalent to in different applications could also be done. For example, the results of most simulations that have been presented in this dissertations were run for around one thousand time slots, but one thousand time slots could be a very long or a very short time for different applications. Although the maximum number of time slots were increased up to ten thousand and the networks remained stable, it is still hard to tell if that is long enough. In general all network parameters could be tested further to see how they impact both policies but it would be better to learn more about IoT network parameters first so that the tests are meaningful and have some direction.

In the production planning chapter, there is no way to stop nodes from only receiving resources and not contributing back to the network. Ideally we would like to build on top of the policies presented in this dissertation by adding some functionality to incentivise nodes to both produce and consume resources as best as they can. Again, more work could be done on production scheduling if we understood more about the production rates of different producers in IoT networks.

7.3 Conclusions

As the number of devices connected to the Internet continues to grow there will be more opportunities for IoT devices to cooperate with one another to complete tasks. This will also increase the need for policies that can allow devices to successfully collaborate with one another overtime. Both the Max-Weight policy and the Average-Weight policy could prove to be two very usable policies as a way of sharing resources between IoT devices to complete tasks while ensuring stability. Much work is still do be done in terms of learning about the production and arrival rates of different IoT devices as it has unfortunately not in the scope of this dissertation. When the number of resource types increases to more than one, producers can cooperate with one another even further by scheduling the production of their resources. Production scheduling ensures that resources are not wasted and that each consumer receives resources that they need most.

Bibliography

[1] K. Douzis, S. Sotiriadis, E. G. M. Petrakis, and C. Amza, "Modular and generic IoT management on the cloud," *Future Generation Computer Systems,* Article vol. 78, no. Part 1, pp. 369–378, 1/1/January 2018 2018.

[2] H. Mora, J. F. Colom, D. Gil, and A. Jimeno–Morenilla, "Distributed computational model for shared processing on Cyber–Physical System environments," *Computer Communications,* Article vol. 111, pp. 68–83, 10/1/1 October 2017 2017.

[3] D. Navani, S. Jain, and M. S. Nehra, "The Internet of Things (IoT): A Study of Architectural Elements," ed: IEEE, 2017, p. 473.

[4] Cisco, "Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are.," ed: Cisco, 2015.

[5] G. Bedi, G. K. Venayagamoorthy, R. Singh, R. R. Brooks, and K. Wang, "Review of Internet of Things (IoT) in Electric Power and Energy Systems," *IEEE Internet of Things Journal, Internet of Things Journal, IEEE, IEEE Internet Things J.,* Periodical no. 2, p. 847, 2018.

[6] H. Green, "The Internet of Things in the Cognitive Era: Realizing the future and full potential of connected devices.," ed: IBM Watson IoT, 2015.

[7] P. Giovanni, C. Mario, and M. Vincenzo, "Bluetooth 5 Energy Management through a Fuzzy–PSO Solution for Mobile Devices of Internet of Things," *Energies, Vol 10, Iss 7, p 992 (2017),* article no. 7, p. 992, 2017.

[8] G. Iosifidis and L. Tassiulas, "Dynamic Policies for Cooperative Networked Systems," *EC: Electronic Commerce,* p. 36, 06/27/ 2017.

[9] L. Tassiulas and A. Ephremides, "Dynamic server allocation to parallel queues with randomly varying connectivity," *IEEE Transactions on Information Theory,* vol. 39, no. 2, pp. 466–478, 1993.

[10] M. G. Markakis, E. Modiano, and J. N. Tsitsiklis, "Delay analysis of the Max–Weight policy under heavy–tailed traffic via fluid approximations," in *2013 51st Annual*

Allerton Conference on Communication, Control, and Computing (Allerton), 2013, pp. 436–444.

[11] C. Bradshaw. (2011, 16/05/2018). *The 6 primary types of Renewable Energy*. Available: https://www.renewablesguide.co.uk/primary-types-of-renewable-energy

[12] L. Georgiadis, M. J. Neely, and L. Tassiulas, *Resource Allocation and Cross–Layer Control in Wireless Networks*. now, 2006.

[13] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks,* Article vol. 54, pp. 2787–2805, 1/1/2010 2010.

[14] S. S. S. R. Depuru, L. Wang, and V. Devabhaktuni, "Smart meters for power grid: Challenges, issues, advantages and status," *Renewable and Sustainable Energy Reviews,* Review Article vol. 15, pp. 2736–2742, 1/1/2011 2011.

[15] E. W. Weisstein. (2018, 19/05/2018). *Complete Graph*. Available: http://mathworld.wolfram.com/CompleteGraph.html

[16] E. D. Kolaczyk and G. Csardi, *Statistical Analysis of Network Data with R*. Springer, 2014.

[17] M. E J Newman, *Assortative Mixing in Networks*. 2002, p. 208701.

[18] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small–world' networks," *Nature,* vol. 393, p. 440, 06/04/online 1998.