Aalto University
School of Science
Degree Programme of Computer Science and Engineering

Axel Eirola

Improving packet transport network efficiency using capacity aware routing

Master's Thesis
Espoo, September 23, 2012

Supervisor:	Professor Heikki Saikkonen
Instructors:	Vesa Hirvisalo D.Sc. (Tech.)
	Timo Olkkonen M.Sc. (Tech.)

Aalto University
School of Science
Degree Programme of Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author:	Axel Eirola		
Title:	Improving packet transport network efficiency using capacity aware routing		
Date:	September 23, 2012	**Pages:**	vii + 64
Professorship:	Software Systems	**Code:**	T-106
Supervisor:	Professor Heikki Saikkonen		
Instructors:	Vesa Hirvisalo D.Sc. (Tech.) Timo Olkkonen M.Sc. (Tech.)		

The role of transport in our society is ever growing with increasing globalisation, and new methods are needed to keep up with demand within the frame of available resources. This means using technological advancements in order to utilise currently available capacity to its fullest. In this thesis we focus on improving efficiency of routed packet transport networks by using routing algorithms that take into account real time network package load data.

Given a transport network that moves packets between stations in the network, we aim to decrease packet costs and increase the throughput of the network. We do this by utilising data on the network load in order to find more efficient paths for the packets to travel on. For this we develop methods and software to simulate and measure different routing schemes, enabling us to compare how the usage of the network load data affects the network performance.

This work resulted in the *capacity reserving* router, which keeps track of network resource usage in order to avoid congestions under heavier loads. This method was able to increase the network efficiency in our simulations by up to 100%, compared to similar routing without capacity reserving. Additionally, the cost of individual packets decreased by up to 50% under heavy loads. These results enable more efficient usage of packet routing in transport networks.

Keywords:	shortest path problem, parcel routing, transport network
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Axel Eirola		
Työn nimi: Jakeluverkostojen parantaminen hyödyntämällä täyttöastetietoista reititystä			
Päiväys:	23. syyskuuta 2012	**Sivumäärä:**	vii + 64
Professuuri:	Ohjelmistojärjestelmät	**Koodi:**	T-106
Valvoja:	Professori Heikki Saikkonen		
Ohjaajat:	Tekniikan tohtori Vesa Hirvisalo Diplomi-insinööri Timo Olkkonen		

Laajenevan globalisaation myötä kuljetusten merkitys yhteyskunnassamme on kasvamassa, ja tätä varten tarvitsemme uusia keinoja tyydyttääksemme tämän tarpeen olemassa olevien resurssien puitteissa. Tämä tarkoittaa teknologisten edistysaskeleiden käyttämistä hyödyntääkseen olemassa olevia resurssia mahdollisimman tehokkaasti. Tässä työssä keskitymme parantamaan reititettyjen pakettikuljetusverkkojen tehokkuutta ottamalla huomioon verkon täyttöastetta reititysalgoritmeissa.

Pyrimme alentamaan pakettien kuljetuskustannuksia ja parantamaan suorituskykyä kuljetusverkossa joka siirtää paketteja verkossa olevien asemien välillä. Mahdollistamme tämän hyödyntämällä verkon reittien täyttöastetta löytääksemme tehokkaampia polkuja joita pitkin paketit voivat kulkea. Tätä varten kehitämme menetelmiä ja ohjelmia joiden avulla voimme simuloida ja mitata eri reititysmenetelmiä, joka antaa mahdollisuuden verrata miten täyttöasteen hyödyntäminen vaikuttaa verkon suorituskykyyn.

Työn lopputuloksena on *tilaa varaava* reititin, joka ylläpitää tietoa verkon resurssikäytöstä voidakseen välttää tukoksia korkeamman kuormituksen alla. Tämän menetelmän myötä onnistuimme parantamaan verkon tehokkuutta simuloinneissamme jopa 100%, verrattuna vastaavanlaisiin reitittimiin ilman tilan varausta. Tämän lisäksi yksittäisten pakettien kustannukset laskivat jopa 50% korkeimpien kuormitusten alla. Nämä tulokset mahdollistavat suoraan tehokkaamman pakettien kuljetusverkkojen hyödyntämisen.

Asiasanat:	lyhimmän polun ongelma, pakettireititys, kuljetusverkko
Kieli:	Englanti

Aalto-universitetet
Högskolan för teknikvetenskaper
Examensprogram för datateknik

SAMMANDRAG AV DIPLOMARBETET

Utfört av:	Axel Eirola		
Arbetets namn:	Effektivare distributionsnätverk med hjälp av kapacitet medvetend dirigering		
Datum:	23 september 2012	**Sidantal:**	vii + 64
Professur:	Programsystem	**Kod:**	T-106
Övervakare:	Professor Heikki Saikkonen		
Handledare:	Teknologie doktor Vesa Hirvisalo Diplomingenjör Timo Olkkonen		

Genom spridande globalisering blir transport en hela tiden viktigare del av vårt samhälle, och nya metoder krävs för att kunna möta kraven inom ramen för tillgängliga resurser. Detta innebär ett behov att använda sig av teknologiska framsteg för att utnyttja existerande kapacitet till sitt fullaste. I detta arbete koncentrerar vi på att höja effektiviteten på packet transport nätverk genom att utnyttja nätverkets kapacitet information i dirigeringen av paketen.

Vi ägnar oss åt att sänka transportkostnaderna för paketen och höja effektiviteten för nätverk som transporterar paket mellan stationer i nätverket. Vi möjliggör detta genom att utnyttja kapacitet information inom nätverket för att finna effektivare rutter för transporten av paketen. För detta utvecklar vi metoder och program för att simulera och mäta olika dirigeringsmetoder, som möjliggör oss att jämföra hur utniyttjandet av kapacitetinformationen inverkar på nätverkets prestanda.

Detta arbete resulterade i en *kapacitet reserverande* paket dirigent, som håller reda på nätverkets resursbruk för att kunna undvika stockningar under högre belastning. Med hjälp av denna metod lyckades vi höja nätverkets prestanda i våra simuleringar med upp till 100%, jämfört med liknande dirigenter utan kapacitet reservation. Däröver sjönk kostnaderna för enskilda packet med 50% under högre belastning. Dessa resultat möjliggör direkt effektivare förbrukning av paket dirigering i transport nätverk.

Nyckelord:	kortaste vägens problem, paket dirigering, transportnätverk
Språk:	Engelska

Acknowledgements

I would like to thank my instructors for their contributions to this thesis project; Vesa Hirvisalo for guidance and wisdom in academic methods as well as writing, and Timo Olkkonen for providing technical and real world assistance in times of need. Additionally, I would like to offer my gratitude to Heikki Saikkonen for taking the time to supervise my work.

In addition I want to thank all colleagues at Trimico for letting me work on my thesis although there would have been more important things to do. Additional thanks go out to Juha for taking upon himself the role as the pusher and providing a general driving force; Ismo for valuable insights in the fields of transport and logistics; and Jyri for providing the original project idea.

Last, but not least, I want to thank my family and friends for providing much needed support as well as distractions from the exhausting problems faced. This includes, but is not limited to: mom, dad, Emil as well as everyone at `#kumikanaultimate` and `#mtpf`.

Helsinki, September 23, 2012

Axel Eirola

Contents

Chapter 1

Introduction

Transport is a fundamental part of the globalised society we live in today. As different stages of production are centralised in geographically distant areas, the transport of goods between these centres becomes a strain on available resources. And as we as a society get more conscious about the availability of our resources, and the sustainability of our way of life, we start to put more weight on the usage of these limited resources. This means that we strive to utilise them to their fullest.

With access to additional information we are able expand the possibilities by which systems and algorithms can optimise their real world performance. The more information a system has of the respective real world state, the more accurately it is able to model it and adapt its actions accordingly. Thus we can usually improve performance and save resources by utilising more of the available data. This is one of the main ideas behind the *intelligent transport* [8] term, which describes the attempt to improve the performance of transport systems through better informed and more coordinated decisions.

What this means is that by building systems which utilise internal as well as external data, we can increase the interoperability between systems or parts of systems to increase overall performance. This is possible on different levels of transport networks, such as cooperation of cars and traffic lights to minimise breaking, or coordination of global shipment routes and warehouse positioning. Everything counts in the pursuit for reduced costs.

In this thesis we focus on the possibilities of increasing packet transport network efficiency by utilising available knowledge of the network state in packet routing algorithms. In short, we provide the routing algorithms of the network with increased information on the network state and study how the utilisation of the added data affects the performance of the network.

1.1 Problem statement

The general problem of routing packets in transport networks is in itself nothing new. The shortest path problem is a basic part of graph theory, with multiple solutions and countless applications in different fields. But while the general problem is well known, the different special cases of the problem are not as thoroughly studied. Therefore the main goal of this thesis is to examine how the performance of transport networks can be improved by providing the packet routing algorithms with additional data on the network state. Specifically, how the effectiveness of the network increases when taking into account information on the capacity load of the connections of the network.

But in order to actually utilise the available data to increase the effectiveness of the network we need to produce and measure specific algorithms that provide the shortest path for the packets travelling in the network. This is where the main work of this thesis lies. The routing algorithms used define the way the data is utilised, and provide a way for us to measure their potential benefits.

As algorithms for the actual shortest path problem are abundant, the real problem boils down to defining what is a *good* path in the context of the network model. This can be further reduced to defining the cost of a path in the network, which will be minimised by the shortest path routing algorithm. This all depends on the model of the network, which needs to be specified in order to give the cost of a path any meaning.

With a model of the network, and the cost of a path in the network, we will be able to define methods for actually executing the model. And based upon these methods, we construct a simulation environment in which we are able to run the implemented routing algorithms. This will enable us to gather measurements of the network performance in different scenarios while using different routing algorithms, which finally gives us a indication of the potential benefit of the used information.

After a final solution for the routing algorithm choice, we still want to assure that the results are usable in practical situations. For this we measure the routing algorithm running times in varying network sizes to see how well they would fare in larger scale production environments.

The main problem of studying the effect of the increased information utilisation is thus divided into smaller tasks. Through the definition of the cost of a path and the model of the network the study finally culminates in the measurement of the performance of the networks with different routing algorithms.

1.2 Contributions

The main contributions of the work done in this thesis include the introduction of a novel routing algorithm based upon traditional shortest path routing algorithms. The *capacity reserving* routing algorithm utilises the capacity load state of the network to route packets around any congestion points in the network, and balance the load over multiple paths with available capacity. This both decreases the cost of individual packets travelling in the network, as well as increases the overall throughput of the network under high packet loads.

In addition to the introduction of the capacity reserving routing algorithm, this thesis presents a model of packet transport networks, along with methods for simulating, measuring and comparing them. The usefulness of these methods extend beyond routing algorithm performance measurements performed in this thesis. The same methods can be directly applied to other tasks such as planning future networks or predicting the behaviour of current ones in specific situations.

1.3 Thesis structure

The rest of this thesis is divided into seven chapters out of which the first four approach the problem of packet routing at incrementally lower levels of abstraction, and the latter three evaluate, discuss and conclude the results produced. Each of the different levels of abstraction aim to give a solution to one or more of the subproblems stated in section 1.1.

We start off in with a high level overview of the background of the fields of transport and routing algorithms in chapter 2. This includes introduction of the basic building blocks of a transport network and the economics around it, giving a context for the costs of transportation. Additionally, a summary of previous work and current state of routing algorithms that we will base our constructive work on.

In chapter 3 we construct a transport network model based on the knowledge from the previous chapter, but one which is specific to the type of packet transport we are focusing on in this thesis. As a part of constructing the model we identify, label and describe the specific active entities in a transport network that will be used in later chapters.

The methods in which we utilise the model from the previous chapter will be introduced in chapter 4. This includes description of the specific ways in which we aim to build, simulate, route and measure the transport network model.

Actual implementation of these methods is described in grater detail in chapter 5. Here we go over the ways in which the network behaviour and the routing algorithms are represented as computer programs used for the simulation of the network model. This is the lowest level of abstraction, and in the greatest detail, the network model and simulation is described in this thesis.

The results of the previous chapters are evaluated in chapter 6. This is achieved by simulating the network activity using different routing methods and comparing the measured efficiency. The measurements are plotted and explained in this chapter. In addition to the network performance, the running time of the routing algorithms are studied to validate their feasibility in practical applications.

In chapter 7 we focus on the implications of the results from the previous chapter. We also take a look at how the methods could be applied into real world situations and what the possible benefits would be.

Finally, this thesis is concluded in chapter 8 with a summary of the work done. Additionally, we explore some ideas of how the research could be continued in possible further work.

Chapter 2

Background

This chapter describes the background for this thesis from two perspectives. Firstly, the field of logistics and transport is introduced in section 2.1 to give the reader a basic knowledge of the application environment of the routing algorithms, and their potential benefits in real world scenarios. Different network structures are also introduced, and the network structure used in this thesis is identified.

Secondly, a brief introduction into the history of shortest path and routing algorithms is given in section 2.2, along with current usages in the field of transport. Additionally, notes on previous work in load-aware routing algorithms are included.

2.1 Transport and logistics

Logistics is the management of resource flows within a company. It is the glue that binds together the main operational areas of production, distribution and marketing; and thus more than simple shuffling of goods [25]. Logistics is an umbrella term for all the effort in a company that is put into having the right resources, in the right place, at the right time. In addition to material management, logistics encompasses areas such as supply chain management, warehouse management and information management [27].

With the help of logistics a company ensures that customer demand is met, production plants are utilised and that warehouses are not filling up, nor sitting empty. With successful logistics the largest possible amount of resources see the largest possible amount of utilisation, giving the company larger profit margins and more revenue. Logistics can account for up to 10% of the total company expenses and actual transport taking up 4%, with values are even higher on a national scale [24].

Transport is a central part of logistics. It deals specifically with the movement of physical goods from point A to point B, and logistics include numerous situations where physical goods need to move between places [20]. For example, getting raw materials from suppliers to production facilities, components from factories to warehouses, and finished products from warehouses to retail distribution. While the different situations all have separate requirements on volumes, durations and costs; they all share in common the fact that they utilise some transport network services to transfer the goods to their destinations.

Transport can generally be divided into two categories: passenger transport and freight transport. The distinction has to be made since there are greatly different requirements for each. For example, while all modes transport naturally aims for a timely delivery, large deviations from timetables are far more critical in passenger transport than in freight transport. In addition to this, both categories usually require separate vehicles and containers for transport, with different density classes and other requirements.

The movement of goods can be achieved by multiple different modes of transport. The most common are flight, ship transport, rail and truck. Other examples include pipes, cable and space transport. The different modes of transport still have a common structure with loaded vehicles travelling between interlinked stations. The set of stations and their interlinked connections form a *transport network*.

It is often necessary to utilise more than one mode of transport for goods to reach their destinations. For example, a shipment between continents usually travels by truck to and from an airport, and by plane between the airports. This is termed *multimodal* transport. The basic structure of the network model is retained, dispite the usage of multimodal transport. Packages still travel form hub to hub along the edges of the network, with different modes of transportation on different edges, and transloading between these occurring at the hubs.

2.1.1 Transport costs

Transport is a field of business in the same sense as any other. Transport companies get revenue from customers using their transport services for their logistics solutions, and they have expenses from the wages, fuel costs and other operational costs. Naturally a company will want to increase its revenues by acquiring more and larger customers, while decreasing their expenses by minimising their resource usage. Increasing the revenues is a marketing problem, but decreasing the transport expenses is in many ways an engineering problem.

We break down the costs of transport to find the metrics that produce the cost expenses for the company. After identifying the metrics, we will be able decrease them. The different costs of transport can be categorised as internal and external costs [13]. In the following, normal business operational costs are not taken into account since they are not directly affected by the actual packet transport, which are the main point of interest in this thesis. The costs identified for transport are:

Internal costs also called market costs, are costs directly included in the price of transport, and which the transport service provider directly pays. These costs include among others:

- **Fuel costs** The cost of fuel required to operate vehicles.
- **Wage costs** Wages for vehicle operators and hub workers managing packet loading and routing.
- **Time costs** Costs for prolonged transport, and delays. Includes costs for maintaining integrity of goods such as security or refrigeration, as well as customer dissatisfaction costs caused by delays.
- **Vehicle costs** Costs related to acquiring and maintaining a vehicle fleet.

External costs are costs directed to parties other than the transport company. These costs usually originate from the use of public sector resources by private sector companies. Such resources include rail and road networks, harbours and in some extent unpolluted air.

- **Pollution costs** Most modes of travel pollute the air in some way, leading to decreased air quality and any complications related to this.
- **Noise costs** Costs caused by noise generated from transport, which inconveniences any nearby third parties.
- **Congestion costs** Usage of road networks lead to increased congestion, which either increase costs for other users of the network, or costs of improving the network throughput.
- **Accident costs** Linked to congestion costs, accident costs increase with the utilisation of the network and is related the safety of the drivers and equipment used.

Naturally, none these costs can easily be decreased without affecting some other cost. For example, decreasing the transport time by having vehicles travel at higher speeds would drive up the fuel costs due to increased consumption at higher speeds. Alternatively fuel costs could be decreased by using larger vehicles which are able to transport a larger amount of goods, but this would again increase the transport time as the vehicles would need to wait until they are at full capacity to bring in savings from their increased size. Because of this interdependency between the different costs, they have to be weighted and balanced to produce a reduced total combined cost.

The costs of transport listed above can each be linked to a metric of a single consignment transported from its origin to its destination. In addition to static costs related to the consignment there are also three distinct variables that affect the total cost of the consignment. Here we assume that the network of transports is already in place, or are accounted into the static costs.

Static cost Fixed costs related to every packet. This include packaging costs, bureaucratic costs and wages for pick-up and delivery workers.

Distance The total distance travelled by the goods affects the costs of fuel and pollution.

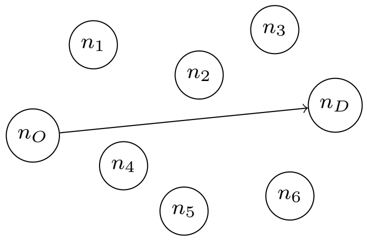
Changes The number of transloading performed during the packet journey affects costs of hub worker wages and station fees.

Duration Costs based on packet delivery time, such as time costs and customer satisfaction.

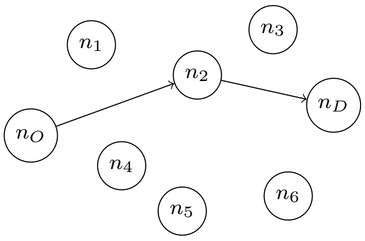
The aim of any transport service is to minimise these metrics, which will directly decrease the costs associated with the transports. The values can be minimised in multiple different ways: by changing the network structure, the fleet configuration, station equipment, path choices, and so on.

Like the costs before, the above metrics are usually linked to each other as well. Taking the absolutely shortest path might not yield the least amount of changes, and the shortest duration might yield a far greater distance. Naturally, these relations are heavily dependent on the network structure.

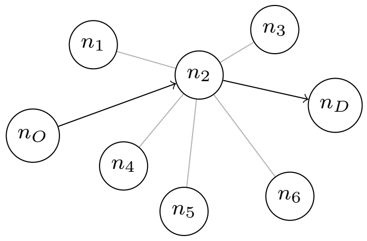
In addition to the costs being directly linked to packet metrics, the choice of path can also affect the expenses of a company in other ways. Since the routing choices of the packets has the ability to affect the total throughput of the network: improving the routing algorithm has the ability to increase the capacity of the network. In practice this would mean increased utilisation, and decreased need for expanding the vehicle fleet capacity.



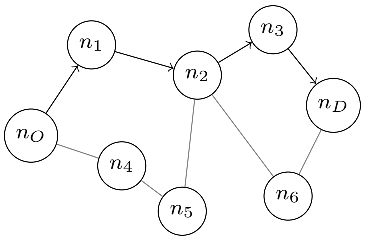
(a) Point-to-point.



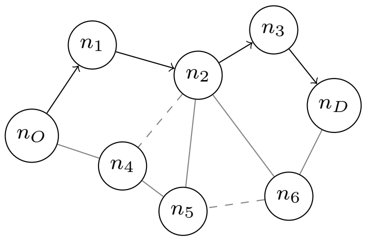
(b) Corridor.



(c) Hub-and-spoke.



(d) Fixed routing.



(e) Flexible routing.

Figure 2.1: Five main transport network strategies.

2.1.2 Transport networks

The multimodal transport networks in use vary largely in their topologies. The network might acquire very different structures depending on the modes of transport used, the level of integration between the modes, and the costs linked to each step in a delivery. Five main network types as defined by Hesse [12] are visualised in figure 2.1 and described below.

Point-to-point 2.1a. Each consignment is directly transported from consignor to consignee without any change of container or vehicle. One could think of taxis and cycle courier as point-to-point network operators.

Corridor 2.1b. Similar to point-to-point networks in the sense that there is a specific path from consignor to consignee, but where there is a need to change vehicle or mode of transport. For example, a truck and ferry combinations.

Hub-and-spoke 2.1c. A network spanning multiple locations. Two types of transports occur: low volume transports to and from hubs, and high volume transports between hubs. Small networks might suffice with a single hub which handles all packets, but larger networks require multiple hubs that are tightly connected with high volume transports. This is how most international transport services such as TNT and UPS operate.

Fixed routing 2.1d. A network created by predefined fixed routes that the vehicles travel. Consignments travel along the routes from station to station, and make appropriate route changes at the stations to reach their destinations. Railway and bus networks usually work in this way. This is the main type of network inspected in this thesis.

Flexible routing 2.1e. Similar to fixed routing networks but with the possibility to flexibly alter the routes according to the demand of the network. This requires a much higher level of integration between the vehicles and the consignment management to efficiently reroute the vehicles to optimal paths.

2.2 Shortest path problem

The problem of finding the shortest path in a network has probably existed as long as there have been roads or other passages for transport. Choosing the right branch in a junction is never a trivial task, and doing it for each junction along the whole path even less so. One of the first formal description of the shortest path problem can be found in Bellmans ‘On a routing problem’ [4].

The methods for finding the shortest path in a graph vary largely depending on how the problem and context is restricted, and what kind of answer is ultimately sought after. For example, finding the shortest path from point A to point B in a unweighted undirected graph can be solved easily with a breadth-first search starting from either point [23, Ch. 5]. But if the edges are weighted, and especially if negative weights are allowed, the methods for finding the solutions become more complex. Additionally, if the required solution is not just the single shortest path form point A to point B, but the shortest paths to *all* points from point A, or even all paths between all points, then the methods applied usually change as well.

The most common scenario is finding the shortest path between two points in a directed positively weighted graph. This is because most naturally occurring networks are these type of networks, and there is usually only a need to retrieve one shortest path at a time. The most common use case for this is any mode of transport, be it transport of tangible goods or abstract information, where there is a specific designated recipient. The best known solution to this specific version of the shortest path problem is Dijkstra's algorithm, published in 1959 [7]. The efficiency of the algorithm was later improved by Fredman and Tarjan in 1984 [9] with the use of Fibonacci heap based priority queues, but the original idea was still retained.

2.2.1 Journey planners

A common extension to the problem is the inclusion of the temporal dimension, which enables the usage of shortest path routers for planing public transport journeys. Constraining the transport to specific points in time changes the problem significantly [6]. This means that instead of having free connections between the nodes, a specific edge between two nodes can only be travelled at specific time, according to timetables that accompany the network. This creates a larger cause-and-effect chain for the chosen paths, where a certain path can look promising in the beginning, but where the traveller would in the end fail to reach a node in time for a crucial departure.

The general increase in algorithmic efficiency and available computational power during the last century has enabled the widespread usage of these kind of routing solutions in public transport. This can be seen in the increased amount of journey planners available in multiple municipal areas. Almost all larger regional transport services provide journey planners for their bus and train networks [15]. And recently even Google has incorporated public transit journey planners into their mapping and directions services.

These journey planners provide greater service than just giving the shortest path for the journey: they also provide the user with the timetable data, which would otherwise be bothersome to manually search to find a suitable path. In this way the shortest path routers appear as tools like search engines that provide information form a network, instead of simply algorithmic solutions to specific problems.

2.2.2 Packet routing

Routing algorithms in the goods transport industry vary heavily on the network type used. Some network types such as point-to-point and corridor do not provide any possibilities for path alternatives, and all packets must follow the same path. Hub-and-spoke networks might provide some routing options on inter-hub transports, but these are mostly statically routed with predefined high-volume transports [20, Ch. 9].

The only network types described here that actually provide the possibility for dynamic packet routing are the networks employing fixed or flexible routed topologies. And even these routed networks may use static routing which is tightly connected to the network design. So the importance of a routing algorithm in travelling a network is dependant on the structure of the network. If the network is very sparsely connected, or if there are only few redundant routes in the network, then the possibilities of improvement in the area of packet routing may be very slim. The smaller amount of possible routes there are in a network, the smaller the significance of the routing algorithm is. These types of networks are on the other hand improved by tweaking the network structure and transport routes to better accommodate the transportation needs of the network.

The actual routing can be performed in different ways, and the methods can be divided into two categories, both with their own benefits and weaknesses:

Manual routing Highly adaptable to unexpected situations, but requires a lot of human resources. Human limitations and errors make it infeasible for large networks, since the capacity for a human to remember or access vast route and timetable data is limited.

Automated routing Automatically routes packets according to consignment data entered into a computer system. This is efficient, but not necessarily adaptable to special circumstances. Routing methods described in this thesis are all automated routing methods. The automated routing methods usually use some alteration of Dijkstra's algorithm to provide a lowest cost path suggestion. The cost may be calculated from anything from distances or durations, to number of mode changes or country borders crossed [19].

2.2.3 Load-aware routing algorithms

Reserving capacity for travelling passengers or goods is not in any way a new idea. The concept of tickets and place reservations have been used since the dawn of public transport. But there lies a difference between only making reservations for departures, and using this information for making intelligent routing choices.

Passengers are more adaptive than physical packets in coping with fully booked departures and transport congestions. They are able to autonomously choose different routes other than fully booked ones, and reroute themselves when they encounter congestion points. But they are not that efficient in searching the large amount of available data on all the routes and departures, and need to instead resort to heuristics in form of instincts.

Capacity reserving, capacity aware, and load-aware are all names used for routing algorithms that are to utilise available information on the current reservations or capacity load of a network. Research around these topics is frequently published in the field of ad-hoc communication networks, where the routing of the communication connections is a dynamic and resource-aware task [2, 17]. But this is largely separate from the field of transport, and has different requirements for the solution to the problem.

The ad-hoc communication networks are not, by definition, suitable for maintaining a centralised whole view of the network. Ad-hoc networks are decentralised networks consisting of loosely connected equally valued peers, preventing the peers form forming a holistic view of the network. This makes them unable to perform optimal per-packet routing, and will only change paths when the active ones start to degrade. It also means that they cannot do proper capacity reserving, and thus not calculate future loads for future departures. Our model of the packet transport network on the other hand aims to retain and utilise this kind of complete view of the network.

Chapter 3

Transport network model

As we saw in the previous chapter, there are different types of transport services, utilising different types of networks to provide the services. Although they all exist for the sole purpose of moving goods, they work in vastly different ways. We should thus choose and describe the one type of network that is specific to the area of routing algorithms handled in this thesis. This lays a the ground work needed to build up the simulations and routing algorithm implementations, which are needed to measure the effects of capacity reserving in the network.

This chapter describes the network and packet models used by the routing algorithms, and provide a framework for the information required for them to efficiently find good paths in an actual network. This means, that the aim of the model is to provide the basic structure that enables packet routing and performance measurement in the network. Anything unrelated to this would increase the model complexity, without contributing anything to the information provided for the routing algorithms.

The model is largely based on physical transportation networks where the vehicles, transporting the goods in the network, have predefined schedules and routes. This could be railway networks where the trains need schedules for track allocation, or ship routes where the harbour time needs to be predefined to decrease congestion. The common factor is that they have individual vehicles, transporting specific amounts of goods, at specific points in time.

The level of abstraction of the network model is set to the packet routing level, to keep it simple enough, and to focus on the main aspect of this thesis. Since the packet routing options in these networks will not directly affect things such as vehicle safety, road congestion, vehicle delays or breakdowns, worker overtimes and so on; the abstraction level was set above the actual driving of vehicles, and the actual transport of packets from one station to another is taken as a given.

Naturally, there are some specific areas that are certainly affected by the packet routings but are not directly included in the model, such as increased fuel consumption on higher packet loads, or periodical congestion on certain routes caused by third parties. These are abstracted away in the metrics for the packet costs described in chapter 2, and the timetables of the network.

Utilising the capacity information for routing requires a specific kind of network. This is a central network with uniform goods, such as standardised units with constant size and weight. Additionally, the vehicles providing transport capabilities to the network need to be static, so that there is the possibility of surplus capacity. The network should thus be a timetable departure type network. This could for example, be a passenger transport network that, in addition to its main purpose, transports other goods; or a network that has such strict network usage that there are predefined departures on the legs, regardless of the amount of goods to transport.

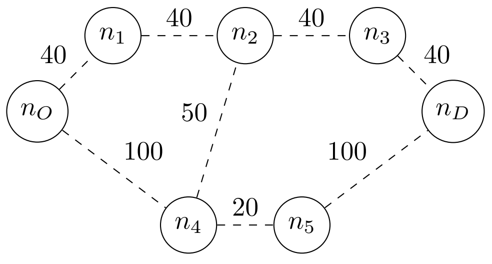
The distinction about what kind of network is used needs to be made, since the routing algorithms would not have any real function in certain types of networks, where the actual network topology already provides trivial paths for the packets. In these types of networks the optimisation is more geared towards restructuring the network and the vehicle routes, in contrast to only affecting the packet path choices, which is an entirely different field.

3.1 Network model

The network model used consists of four different entities. *Stations*, which are the hubs of the network that connect to each other, and to the world outside the network. *Routes*, that connect the stations to each other. *Timetables*, which define at which times the routes are driven. And lastly, *departures*, which are single entities in the timetables, representing a single trip taken by a vehicle. The different entities are described in greater detail below. An example network based upon this model can be seen in figure 3.1.

For simplicity, the network in the figure only contains routes with one leg. In the actual networks used in the simulation, routes with multiple legs are used. One could imagine additional intermediate nodes along the edges, which would constitute to a single route with multiple legs.

The network model does not attempt to directly represent a specific type of transport network seen in real life. The specific criteria is that the departures always travel at specified times, and not in an ad hoc way. This means that model can represent any network with strict schedules, such as bus networks, railway networks or ferries more accurately than lorry or taxi delivery networks which usually use dynamic routing which adapts to the



(a) Network. Dashed lines represent connections, and numbers their distances.

From	To	Dep.	Arr.
n_O	n_1	10:00	11:00
n_O	n_4	10:00	12:00
n_1	n_2	12:00	13:00
n_4	n_5	12:15	12:45
n_4	n_2	13:00	14:00
n_5	n_D	13:00	15:00
n_2	n_3	14:00	15:00
n_3	n_D	16:00	17:00

(b) Timetable.

Figure 3.1: Example network.

current transport requirements.

The model assumes that the state of the network is known at all times by a some central authority. This means that all stations, routes, departures, timetables, packets positions and capacity loads are usable by the routing algorithms when searching for the cheapest route. With the widespread usage of delivery tracking [14], this is in no way an unreasonable requirement. If real-time information about packet positions and departure capacity usages are not available to the routing algorithms, then they can effectively be estimated by keeping track of reserved slots on each coming departure, according to the incoming packet routing requests.

Since the packets in a packet transportation network are not able to move by themselves, without travelling on a specific departure, the model is limited to connected networks to operate properly. This is in contrast to many journey planners, where the passengers being routed are able to take themselves to nearby disconnected stations, albeit at an increased costs.

3.1.1 Stations

Stations are the basic building blocks of the network. They are the interface to the real world, meaning that anything entering or leaving the network does it through a station. Stations are also at the centre of travelling within the network, they are the hubs where anything travelling in the network passes through.

If one considers the underlying graph of the network, then a station would be represented by a node. On the other hand in a real world situation the station would roughly equal a bus stop, a train station or a ship harbour.

The network size is usually represented as the number of stations in the network. The more stations the network has, the larger it usually is, both in a geographical sense, as well as in a functional sense.

3.1.2 Routes

In order to connect stations to each other, the network contains routes. Each route travels through multiple stations, stopping at each one to let any traveller hop on, or off, the route. Routes are directional, meaning that if one can travel in a certain direction in the network, then it does not directly imply that one can travel in the opposite direction.

As each route can have multiple stations, the route is divided into multiple *legs*, where each leg is the part of the route that goes from one station to another. The number of legs in a route is the route *length*. Each leg has a distance and a duration. These are defined by the geographical locations of the stations in the network.

Continuing with the graph analogy, a route would be represented by a chain of edges in the graph. And a leg of a route would be represented by an edge in the graph. In real life the route could be a bus line, a ferry line or a railway line. In a ferry example the route would probably only have one leg though.

The number of routes and their lengths affect the *connectedness* of the network. The more routes there are, and the more legs each route has, the more edges there will be in the underlying graph. This means that anything travelling in the network has more alternatives for finding paths in the network, which increases the possibility for shorter paths.

3.1.3 Departures

Since the routes only specify the lines along which travellers in the network can travel, we need some entity to perform the actual transport. For this we introduce departures. They are instances of generic vehicles that move travellers in the network. Each departure has a route along which it travels, and a starting time at which it leaves from the first station. Each departure also has a certain capacity, which represents the amount of cargo it can transport at a time.

The graph analogy does not extend into the realm of transport networks and departures. In the real world a departure could be a bus departure, or any other vehicle departure.

3.1.4 Timetables

As the network runs on a predefined schedule, the routes need a timetable to specify at which times the departures leave, and arrive, at the stations. For this, each route has a timetable which defines at which times the departures occur. The timetable contains all the departure and arrival times of all legs on a route. Using the timetable the travellers, or routing algorithms, can estimate the times at which they will be arriving at their destination.

In the scope of this thesis the timetables follow a daily cycle, with departures leaving at the same times of the day every day. This could, if needed, be extended to weekly or monthly cycles, but would not give any real benefit to the model in our usage.

3.2 Packet model

Once we have a network model in which we are able to transport goods, we need to specify what the goods are and, how they behave. In our model we use a single goods type, called a *packet.* Each packet has an *origin* station at which it enters the network, and a *destination* station where it exists the network. The packet only exists in the network during its journey between these two stations, and is removed after arrival at its destination.

For the packet to get from its origin to its destination, it must travel using the route departures provided by the network. The packet starts by choosing a route at the origin station, and getting on the next free slot on a departure of that route. If the next departure is full, then the packet needs to wait for the next departure. When the departure arrives at the appropriate station the packet leaves the departure and is *transloaded* to the next planned route, and waits for the next departure on that route. This continues until the packet arrives at its destination.

Each packet consumes one unit of capacity in the departure it is travelling on. If the capacity of a departure is full, then no more packets are allowed to enter the departure until some packets have left the departure. The capacities of the departures, along with the connectedness of the network, together define the total capacity of the network.

The set of routes a packet is loaded to, and the stations it leaves on, defines the *path* the packet takes from origin to destination. The choice of path is specified by the *routing algorithm* used to find the best path. The different routing algorithms are described in more detail in section 4.3.

From the different types of packet transportation network models available, this one was chosen to provide a versatile environment to study the effect of the routes on the network performance. This type of routed packet network model gives the packet routing algorithm enough freedom of choice to be able to provide a measurable change in the efficiency. This was a conscious choice, to bring out the largest potential utility of the capacity information.

Other models, such as those without timed departures, would have simplified the network to a point where the cost function would become trivial. Additionally, the addition of capacity information usage would only have transformed it to a maximum flow problem [23, Ch. 6]. The timetables and departures gives a depth to the problem, in which the capacity information provides a larger possibility of increased performance.

Chapter 4

Applying the model

In this chapter we introduce all the methods necessary to implement and measure the effects of the routing method improvements introduced in section 1.2. First, in order to make any measurements at all, we require networks and a simulation context in which to apply the routing algorithms. These are, based on work in chapters 2 and 3, described in sections 4.1 and 4.2. After this, section 4.3 takes a more formal look at the different routing algorithms used in the final simulations and measurements. Section 4.4 focuses on what kind of data is available in the simulations, and how we can use the data to measure the performance of the routing algorithms.

4.1 Random networks

For us to be able to execute the planned simulations, we need a set of networks to run them on. These could either be gathered form public available data of real world networks, or by generating them randomly. The problem with the real world examples appeared to be a too small amount of networks, with too inconsistent structure for reliable simulation. For the measurements in this thesis we chose to use randomly generated ones, in order to have a large enough set to run the simulations on. Additionally, random networks enable us to more efficiently find networks with specific desired properties. Since there is no 'average' network to simulate, we chose to run simulations on multiple networks and average the results.

The networks used in the simulations is based on the model form section 3.1, and consist of stations, routes, and departures. The packets move from station to station using departures that travel on predefined routes between the stations. Each route consists of legs between stations, with each leg having a duration and a distance. Since the network of stations is not fully connected, packets travelling between stations usually need to change to other departures at some point in their journey. This transloading takes a given amount of time, specific to the station where it is performed.

As the network structure plays a large role in the way packets can be routed within them, it is crucial to use real-life-like network models for constructing the random networks. To achieve this, we decided to use the Barabási–Albert model [3] to produce scale-free networks. The advantage of these networks is that they contain nodes of varying sizes with a degree distribution following the power law. This means that the resulting network will contain both highly connected, and sparsely connected nodes, just as there are large and small cities. The model has been used previously by Berliant [5] for modelling transport networks. The actual routes of the transportation network are generated as random walks in the network produced by the Barabási–Albert model.

The routes of the network travel at a daily cycle, with a certain amount of departures per day, and the same timetable repeating for simulations longer than one day. This enables the simulation to model the day and night rhythm of departures and network capacity. The day part of the cycle has more and faster connections, while during ht night the packets will usually be waiting for next day departures.

The characteristics of the network affects the ability of the routing algorithm to improve based on the usage of capacity information. For example, the connectedness of the network plays a large part in this, as it enables more possibilities to reroute packets around congestion points in the network.

Additionally, the amount of surplus capacity in the network limits the usability of the possible performance increase. If no departures are near saturation, then there is no loss form it. This means that no improvements can be made by the information of departure capacity loads, since they do not matter with abundant capacity.

The models and methods for building these networks are chosen to provide a good enough scenario for the router to act upon, in order to give an indication of the potential benefits. This is because using a network without any possible variations wouldn't give us any information as to how the capacity reserving router might affect the network.

The topology of the used network still resembles that of networks occurring in the real world. Using uniform random networks could provide even better chances for the router to thrive, as the connections would be more evenly spread, but wouldn't reflect the real world applications with the same accuracy. The usage of the Barabasi-Albert model ensures that the networks will act in natural ways, and that the results from the thesis will be close to the real world.

When generating a random network such as the ones in this thesis, one needs to specify different parameters which affect the resulting networks structure. Most parameters are defined as, usually normally distributed, random distributions so that each station and route is generated an own separate value. In these networks they are:

Network size The amount of stations that will be present in the network.

Edge count The amount of edges per station, along which routes can travel.

Routes per station The number of routes originating from each station.

Stations per route The length of each route in number of stops. The number of routes per station and stations per route defined the total amount of edges in the network, and thus the connectedness of the network.

Number of departures The amount of departures on each route per day.

Departure time The time when the departures leave from their first station.

Route capacity The number of packets that each departure can transport at a time.

Transloading duration The amount of time it takes for a packet to change from one route to another at a station.

Station size An abstract value for the size of an station, used as an weight when generating values such as transloading duration and routes per station for the station. Originates from the degree of the node in the Barabási–Albert model.

4.2 Simulating networks

4.2.1 Discrete event simulation

In order to simulate the transport network, we need to handle the events occurring during the simulation, and record the effects of these events. Transport networks are largely event based, meaning that the state of the network changes only at specific points in time; such as a departure leaving, or delivery of a packet. Thus, we can efficiently simulate transport networks using discrete event simulation [16].

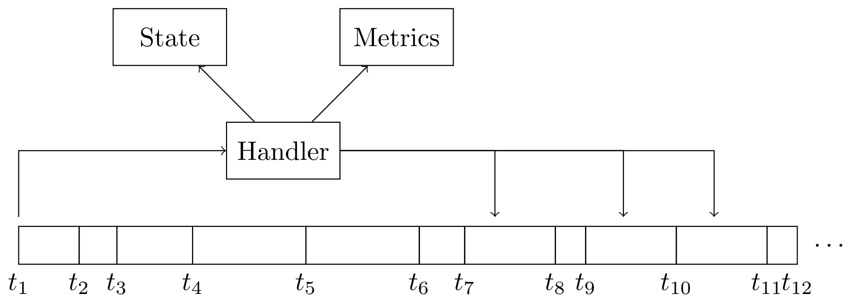


Figure 4.1: Schematic of a general discrete event simulation.

In a discrete event simulation we maintain a simulation clock and queue of events that contains all the upcoming events scheduled to happen later in the simulation. As long as there are events in the queue, the event handler takes the first event in the queue, processes it, and depending on the type of the event and the sate of the simulation, may add any amount of new events into different positions of the event queue. After processing one event, the clock is incremented to the next event in the queue which is then processed, and so on, until there are no more events, or the simulation is otherwise determined to have ended. Usually most simulations end at a predefined point in time, but it might also end after some specific pre-existing input data has been handled. A simple schematic of a general discrete event simulator can be seen in figure 4.1.

Other types of simulations, such as continuous simulation, are also possible, but these provide abstraction levels which might be unnecessary complex for our needs. A continuous simulation could, for example, collect the exact locations of the departures on the paths at each point in time, but this would not exactly give any valuable information in the simulation results, and would thus just introduce unnecessary complexity.

For the simulation to give value to its user, it must output some data on the simulation. Thus the event handler stores metrics on the events it processes during the simulation. This simulation data will then be read and processed after the simulation ends into an report for analysis by the users. The data measured can range from timing execution of processes, to frequencies of certain events, to the complete event trace of the simulation run. All depending on what information is relevant to the specific case at hand.

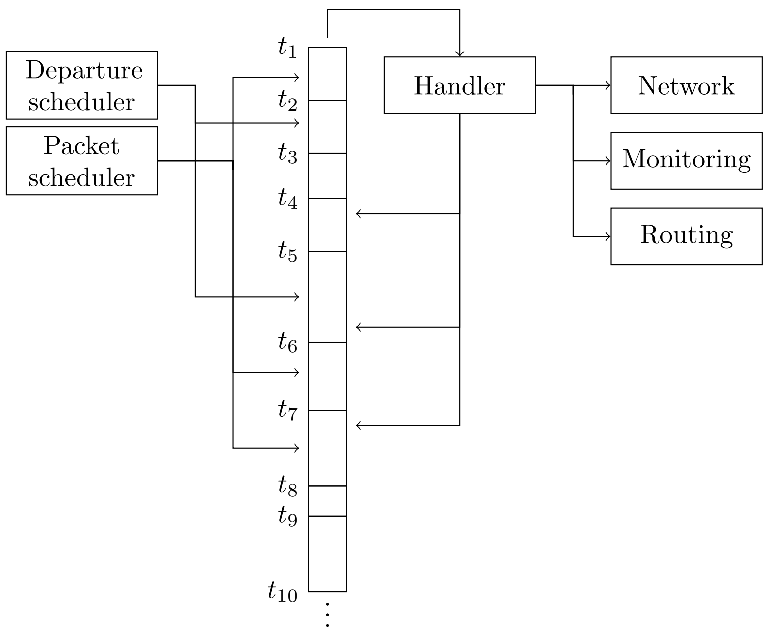


Figure 4.2: Schematic of a transport network simulation.

4.2.2 Simulation of transport networks

For the simulation of transport networks we require a more specialised simulation model. We extend the model in figure 4.1 with dispatchers that populate the event queue with events for incoming packets and departures. These events are generated as the simulation runs based on the network and timetables used for the simulation. Additionally, the simulation needs to be attached to a network which stores the state of the simulation, and to the routing algorithm that determines the paths the transported packets take. With these additions we arrive at the model seen in figure 4.2.

The events processed by the handler are thus those of departure and packet movements. On departures events the handler signals any packets on the station waiting for the departure to board it, and on arrivals signals any packets heading for the destination to leave the departure. For events on packet departures and arrivals, the handler transfers the packets forward to the next routes and stations. All this information is stored in the network to keep the state of the simulation up to date. Additionally, the next events of the packets and routes are added to the event queue to be processed at the specified time.

On packet departures and arrivals, the handler also stores information of the packets to the monitoring module. For each arrival and departure, information about the current packet amounts in the system is updated to enable analysis of capacity usage and network load in the user output. Additionally, after a packet arrival at final destination, data on the completed path of the packet is recorded. This includes data such as duration, distance and the amount of route changes performed by the packet.

Simulating a network by itself, without any packets travelling in it would not give us any benefit. So for each simulation we must, in addition to specifying the network, also specify the packets that will be travelling in the network. The packets in the network model originate at stations in the network, with random stations as their destinations. The packets in the simulation are randomly generated during the simulation, at specified intervals at the stations, emulating customers dropping packets off for transport. Larger stations generate more packets, so each station is given a packet load as average packets per hour, depending on the station's size.

Although the departures have a daily cycle, packets are generated at a steady rate during the whole day. This might not be a realistic case for packet loads originating from parcel services like postal services directed towards consumers, where there are larger peaks during the day. Instead, it behaves more like a production facility generating a constant stream of finished products that need to be delivered as fast as possible through the network.

It would have been interesting to use actual packet data from some transport service provider, but this data is unfortunately treated as company property and is usually not freely distributable. Additionally, in the same way as there are no average networks, there are no average packet loads, meaning that we are better off running multiple simulations on multiple packet loads.

4.3 Routing algorithms

Packets travelling in the network usually have multiple paths to choose from, some of which are arguably better than others. Metrics are calculated for each packet in order to measure the 'goodness' of the given path. The most important aspects of the routes are the amount of changes, the duration and the distance; usually in that specific order of importance. Packet routing, performed by the routing algorithms, is the process of finding a cheap path for a packet destined to travel from its origin station to a destination station. This packet routing problem, based on the model in section 3.2, is a variant of the shortest path problem introduced in section 2.2.

In this section, we describe three routing algorithms that are compared for their performance in packet networks. They all share the same basic principle of building a partial shortest path tree, and choosing the path with the lowest cost. They thus differ mainly in the way they calculate the costs of the path alternatives, which is what actually defines their behaviour. The cost function for the algorithms takes three parameters: distance travelled, changes made and time taken. From these values it calculates a single cost for the path alternative. We denote the cost function as $c(d, c, t)$.

The shortest path router is used as a baseline router. The weighted cost router is used as a representation of the widely used routing methods of today [21]. And the capacity reserving router introduces a novel method of routing the packets by utilising capacity load information from the network.

All of the algorithms are optimal in their own sense that they find the path with the cheapest cost in the network. This means that they find the best path according to their own specification of the cost. This does not necessarily mean that the paths they find lead to optimal performance of the network, since the specification might not reflect the overall behaviour of the network correctly.

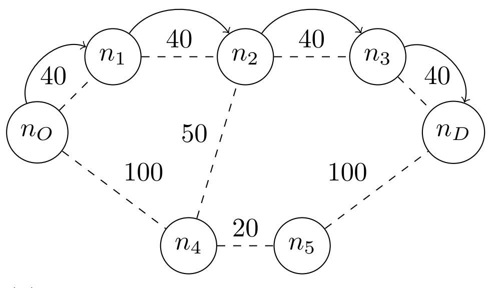
4.3.1 Shortest path

When pondering about travelling from a place to another, one usually comes to the first conclusion that the shortest path is best path. The shortest path minimises the total distance of the journey; and assuming constant velocity, also the travel time. The shortest path routing algorithm does this, and simply finds the path from origin to destination that leads to the shortest distance. It does not take into account any timetables, nor sate of the network. The cost function could then be defined as:

$$c(d, c, t) = d$$

In figure 4.3 we have created an example network containing transport lines between stations, and a timetable of the departures similar to the one in figure 3.1b. The packet is scheduled to travel from the origin node n_O to the destination node n_D. We can directly see that there are three paths available from origin to destination: the top path $(n_O, n_1, n_2, n_3, n_D)$, the bottom path (n_O, n_4, n_5, n_D) and the middle path $(n_O, n_4, n_2, n_3, n_D)$.

The shortest path router would suggest the packet to take the top path, since the accumulated distance is the shortest. The path has a distance of 160, 3 changes and a duration of 7 hours giving a total cost of 160. As we can see it does not take into account the amount of stops in between, nor the slow timetable of the transports. One might argue that this is not the best path in the network, since it takes too long, and has too many changes.



(a) Network. Dashed lines represent connections, and numbers their distances. Arcs represent the chosen path.

From	To	Dep.	Arr.
$\boldsymbol{n_O}$	$\boldsymbol{n_1}$	**10:00**	**11:00**
n_O	n_4	10:00	12:00
$\boldsymbol{n_1}$	$\boldsymbol{n_2}$	**12:00**	**13:00**
n_4	n_5	12:15	12:45
n_4	n_2	13:00	14:00
n_5	n_D	13:00	15:00
$\boldsymbol{n_2}$	$\boldsymbol{n_3}$	**14:00**	**15:00**
$\boldsymbol{n_3}$	$\boldsymbol{n_D}$	**16:00**	**17:00**

(b) Timetable. Bold rows represent the used routes.

Figure 4.3: Shortest path router behaviour in a sample network.

4.3.2 Weighted cost

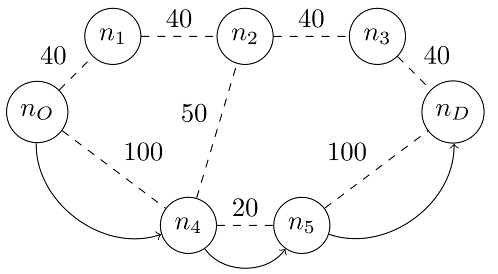
The shortest path routing seemed to be a bit too focused on taking the absolutely shortest path, and did not see the big picture. The shortest path might be the best one in some scenarios, but in the general case it is seldom the fastest. A person travelling the network would not intuitively choose the absolutely shortest path, but would instead take into account the number of changes and departure timetables. And since we have this information at hand, we should definitely use it in some way.

Using the cost breakdown and metrics identified in section 2.1.2 we can construct a cost function of the packet in the form $c_t = c_s + d * c_d + c * c_c + t * c_t$. Here the cost for each individual packet consits of the static cost, distance, duration and number of changes. The static cost cannot be directly affected by the path choice, so this leaves the three metrics left to minimise.

Combining the information on distance, number of changes and path duration we can produce a smarter algorithm that behaves in a more humanly logical way. The weighted cost router takes as parameters weights for the three different metrics; distance, changes, duration. It uses them to calculate a cost for each path as the weighted sums of the values. The cost function is defined as

$$c(d, c, t) = d * w_d + c * w_c + t * w_t$$

The weights given to the algorithm thus heavily alter the behaviour of the algorithm. For example, setting w_{chan} and w_{dur} to zero would give us a shortest path router. The fine tuning of the parameters is a task in itself and depends on the nature of the network and the real world cost of the respective resources. The algorithm could also be modified to include non-linear costs for the metrics to introduce *deadlines* or *limits* after which costs increase at a higher rate. In this thesis we will use the weights $w_d = 0.01 \frac{1}{km}$, $w_c = 2$ and



(a) Network. Dashed lines represent connections, and numbers their distances. Arcs represent the chosen path.

From	To	Dep.	Arr.
n_O	n_1	10:00	11:00
$\boldsymbol{n_O}$	$\boldsymbol{n_4}$	**10:00**	**12:00**
n_1	n_2	12:00	13:00
$\boldsymbol{n_4}$	$\boldsymbol{n_5}$	**12:15**	**12:45**
n_4	n_2	13:00	14:00
$\boldsymbol{n_5}$	$\boldsymbol{n_D}$	**13:00**	**15:00**
n_2	n_3	14:00	15:00
n_3	n_D	16:00	17:00

(b) Timetable. Bold rows represent the used routes.

Figure 4.4: Weighted cost router behaviour in network.

$w_t = 1\frac{1}{h}$. This divide of costs means that one hundred kilometres of distance costs as much as one hour of journey time, in addition to the time taken to travel the distance; and that the change of route costs as much as two hour wait. These values are not directly based on hard data from actual transport networks, but are based on observations of the general function of transport networks. The actual values of the cost function neither directly affects the performance of the routers, as long as they are within feasible limits.

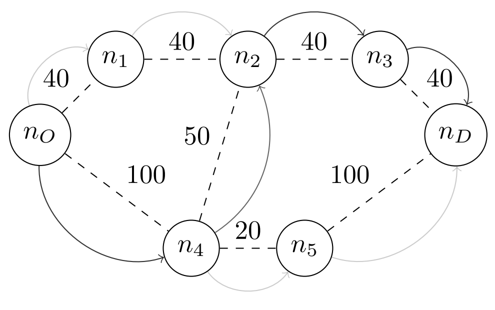
This routing algorithm is similar to how most of the journey planners such as the HSL Reittiopas [11] work. Although they differentiate the distances travelled by foot from those travelled in transit. But in the same way they optimise over both the duration and amount of changes.

Continuing with the same example network and timetable the weighted cost router would suggest the lower path, as seen in figure 4.4. This path has a distance of 220, 2 changes and a duration of 5h totalling in a cost of 11.2. It is longer in distance, but shorter in duration compared to the path suggested by the shortest path router. In most situations this is considered an improvement.

This behaviour makes the weighted cost router sacrifice a short distance, in favour of shorter duration and smaller amount of changes. That means that the router balances the load of the packets away from the shortest paths in the network, and spreads them over the shortest durations.

4.3.3 Capacity reserving

The weighted cost router already provided acceptable routes in the network. But what would happen if the packets the algorithm has ordered to route cannot all fit into the scheduled departures. Luckily, the router is able to know how many packets are travelling on which departure, since the router



(a) Network. Dashed lines represent connections, and numbers their distances. Arcs represent the chosen path. Arc opacity represents the amount of packets on the route.

From	To	Dep.	Arr.	Cap.
n_O	n_1	10:00	11:00	20
$\boldsymbol{n_O}$	$\boldsymbol{n_4}$	**10:00**	**12:00**	**100**
n_1	n_2	12:00	13:00	20
n_4	n_5	12:15	12:45	20
$\boldsymbol{n_4}$	$\boldsymbol{n_2}$	**13:00**	**14:00**	**80**
n_5	n_D	13:00	15:00	100
$\boldsymbol{n_2}$	$\boldsymbol{n_3}$	**14:00**	**15:00**	**80**
$\boldsymbol{n_3}$	$\boldsymbol{n_D}$	**16:00**	**17:00**	**80**

(b) Timetable. Bold rows represent the used routes.

Figure 4.5: Capacity reserving router behaviour in network.

is the one ordering them there. If the router keeps track of the amount of packets routed on each departure, and knows the capacities of the departures, the router can conclude when a specific departure is full. And when a departure is full, it can divert the packet onto another route to avoid having the packet wait for the next departure.

Apart from *reserving* seats on the departures for the packets, the capacity reserving router works in the same way as the weighted cost router. It has the same cost function, and routes the packet onto the exactly same path until any of the departures start filling up. After this point, the packets are routed along lines that still have sufficient capacity available.

Reserving capacity on a departure is nothing new; air planes, ships and trains use this daily to when selling tickets. The novel part is that this information, be it either from tickets sold or any other reservations made beforehand, is used to route new packets to faster places. Of course passengers are able to foresee these kinds of problems, and will find their own ways around a network; but parcels set on a specific path might not be.

If the public transport journey planners provide easy access to the timetable and route data. Then, similarly, the capacity reserving routing provides the same easy access to the current capacity situation.

Once again, using the same example network and timetable as previously, we see how the router works. But this time, let us assume that the transports form n_O to n_1, and from n_4 to n_5, can only carry a small amount of packets compared to the other routes, thus causing bottlenecks in the system. The reserving router notices these bottlenecks when the transports capacity fill up, and use other routes to bypass the bottlenecks.

In the example, the first 20 packets will take the same bottom path as the weighted cost algorithm. When it the bottom path is full the next 20 packets will take the top path, after which the next 60 packets will take the middle path, although it would otherwise be the costliest. In this way, the algorithm is able to have all packets delivered on time, without having any packet wait for later departures. The algorithm thus utilises the maximum available capacity to route packets from origin to destination.

The capacity reserving routing algorithm finds the maximum flow of the network [23, Ch. 6], form the origin to the destination. This of course differs from the original maximum flow problem, in that there are other packets travelling in the network, and they do so on timed departures. But the idea of the solution is similar nonetheless.

4.4 Measuring performance

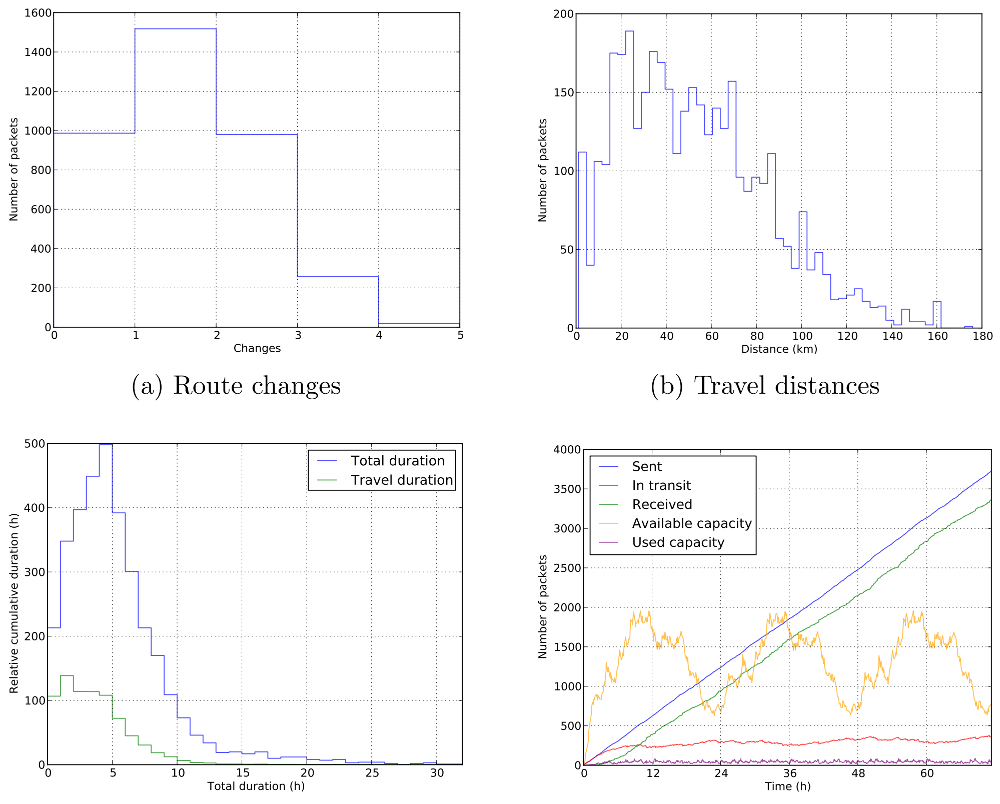
This section describes the methods used for measuring the performance of a transport network simulation, and comparing the performance between different simulation runs. First, in section 4.4.1 we look at the specific measurements we want to use to compare the performance, and in section 4.4.2 we look at how we should go about in comparing the performance.

4.4.1 Available data

To get a view of how the system performs during a simulation run we gather data during the simulation. For the transport network simulations we are mainly interested in the metrics relating to the packets travelling in the network. In order to see how the different metrics from section 2.1.1 behave in our simulated environments, we ran an example simulation with a small network and packet load using a weighted cost router described in section 4.3.2.

Here, we are mostly interested in the paths chosen for the packets, and the way they affect the cost of the packets; as well as, the efficiency of the network. An overview of the paths taken by the packets can be seen from measurements of the distances, duration and amount of changes in the paths. The behaviour of the network as a whole is visible from various packet counters during the running of the simulations.

Figure 4.6 contains visualisations of the data collected. In the figure we can see the amount of changes the packets have taken, the distances they have travelled and the durations taken, as well as the amounts of packets in different states in the system.



(a) Route changes (b) Travel distances

(c) Transport durations (d) Packet amounts

Figure 4.6: Example simulation data

Changes In figure 4.6a, we can see the amount of changes taken by the packets during the simulation. Most of the packets find a path to their destination with one or two routes, while all packets reach their destination in at most five departures. The distribution of the number of changes is a direct result of the Barabási–Albert model the network is based upon, which maintains short paths between most nodes.

Distances The distance of the packets travelled is seen in figure 4.6b. The distances follow a similar pattern to those of the number of changes, partially because the number of changes required correlates to the distance driven. The fluctuations at around the 50, 100 and 150 km marks are directly due to the network structure and packet flow, where the routes of the network have specific distances and specific routes are more heavily travelled. This distance histogram is in a way of signature of the network structure.

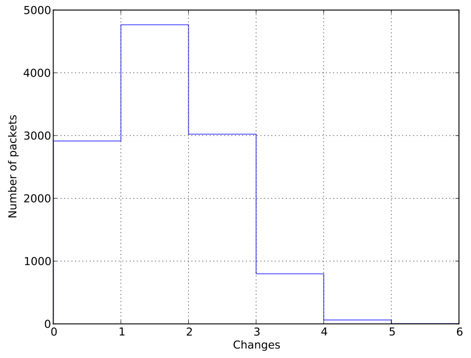
Durations Figure 4.6c on the packet transport durations might require some additional explanation on how to interpret the plot. It resembles a histogram, with the x-axis divided into bins. The x-axis represents the total duration of the package, from entry into the system to the delivery of the packet. The y-axis represents the cumulative duration of all packets that fall within a specific bin, divided by the total duration of the packet. For the blue line of total duration, this means that the value on the y-axis directly equals the amount of packets with the specific duration. For the travel duration, it represent the ratio of the total time spent travelling.

From the plot we can see that the travel duration closely resembles the distance travelled, mainly because the departures travel at relatively constant speeds. The waiting duration of the packets on the other hand is noticeably larger, which means that the packets use a large part of their time waiting for their next departures, which is directly affected by the number of routes and departures in the network, as well as the weights of the routing algorithm.

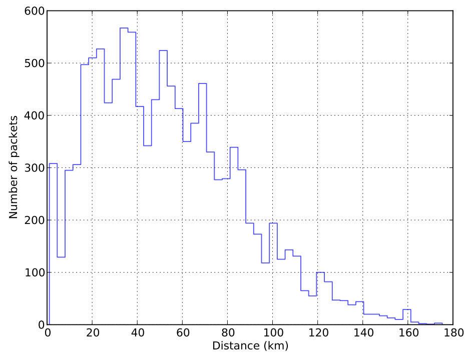
Amounts In figure 4.6d we can follow different packet counters over the duration of the simulation. In blue we have the total amount of packages that have entered the system, which is directly affected by the number of packets generated by the simulation. In red we have the amount of packets that are still on their way to their destinations, which should be low in a well-functioning system. In green we have the amount of packets that reached their destination, which should be steadily growing with the amount of sent packages. Orange plot represents the amount of capacity provided by the departures currently driving; notice the daily cycle. And finally, in purple we have the capacity used, which means the total amount of packets in all departures travelling from station to station.

Notice the *warm-up* time during the first 12 hours of the simulation, during which the departures and packets find their normal daily pace of movement. It is crucial to identify the length of the warm-up time in analysis, and take care that it does not affect any decision making.

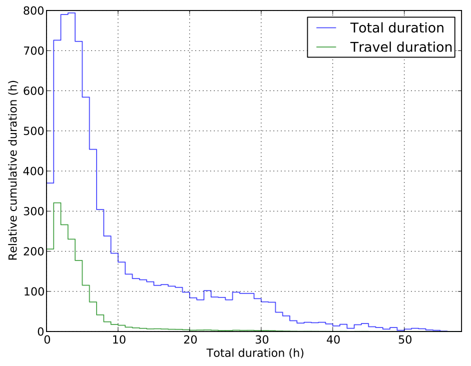
In addition to the warm-up time, there is also a *cool-down* time. This time represents the duration which it takes for packets already travelling in the network to reach their destination. This is important as duration data can only be reliably measured for packets that have reached their destination.



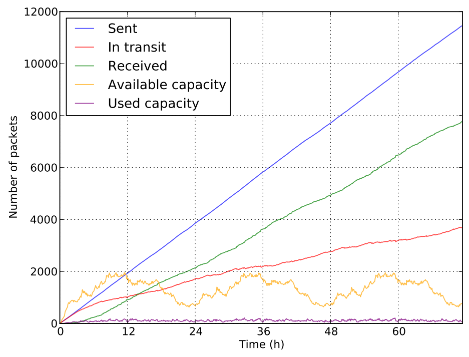
(a) Route changes



(b) Travel distances



(c) Transport durations



(d) Packet amounts

Figure 4.7: Example simulation data for an overloaded system.

4.4.2 Comparing performance

To compare performance between simulations we need another simulation run. For this we took the same network, but with a larger amount of packets generated into the system. Figure 4.7 contains the results from this simulation.

Comparing the results visually we can see that the number of changes and distances of the packets have not changed remarkably, since the router used for the simulation does not take into account the load of the system, meaning that all packets will be taking the precisely same path as in the previous simulation.

The durations of the packets, on the other hand, have clearly increased in the second simulation. This is due to the fact that the increased amount of packet has filled the capacity of the departures, causing some packets to have to wait for the next departures.

From subfigure 4.7d we can see the most dramatic difference between the two simulations, which is that the amount of packets in transit is steadily increasing during the simulation. This is an indication of an overloaded system. In order to measure this inability of the system to handle the packet load, we introduce a *sustainability* metric of the system. The sustainability measures the ratio of packets entering the system to packets exiting the system. So a value of 1 means that the same amount of packets are entering and leaving the system, a sign of a sustainability. Any value larger than 1 means that there are more packets entering the system than what the system is capable of handling, meaning that departure queues, along with the delays, are growing during the measurement interval.

With the help of the sustainability metric, we can measure at what point the network is saturated and cannot transport as many packets as are entering it. Using this measure we can then get a value for the efficiency of the network, by measuring over multiple simulations at which load the network will saturate. A network that can handle a larger amount of packets has a higher throughput. And a packet router that increases the network throughput is more efficient, compared to one that produces a smaller throughput for the network.

As the basic distribution of the values are rather similar in both network simulations, we do not need to include the distribution information in the data comparisons. So to compare performance, we do not necessarily have to include as much data per simulation as in the previous figures. Instead, we can simply use the average values of the different measurements and compare them instead. In this way we get more easily comparable results, as can be seen in table 4.1.

Measurement	Example 1	Example 2
Changes	1.2	1.2
Distance	115	117
Duration	8.1	13.3
Sustainability	1.0	1.6

Table 4.1: Averages of metrics for example simulations in figures 4.6 and 4.7.

Chapter 5

Simulation implementation

Chapters 3 and 4 introduced a model and methods of how a transport network, along with routing algorithms, would work. This chapter aims to provide a way to validate these methods by describing proof-of-concept implementations of these, which will be simulated later in chapter 6.

This chapter is split into two sections: section 5.1 explains the implementation of the simulation methods of section 4.2, and section 5.2 describes the actual routing algorithm implementations.

The simulation framework and routers described here were built from ground up, as custom software specially for the measurements in thesis. While there are well suited network simulation and routing software available under public licenses, we nonetheless opted to use self-made constructions instead. This was motivated by easy access to internal components of the simulation, in order measure different metrics during the development of the simulator. This way, we also get a more practical overview of the behaviour of the constructed system. As it turned out,this became valuable knowledge in planning and performing the evaluations in the next chapter.

All custom software was implemented using the Python programming language [26]. It was chosen for its power of expression, lightweight environment, extensive libraries and fast development pace, making it especially handy for scientific proof-of-concept work. Additionally, the availability of the practical SimPy simulation framework weighted positively in Pythons favour. Both of these factors made the original choice of constructing the simulation by hand an easier job than originally anticipated.

All software described in this chapter, and used in later chapters, can be requested by contacting the author. The software is owned by Trimico Oy, and is not freely distributable.

5.1 Simulator implementation

The basic simulation framework used to implement the transport network simulation is provided by the SimPy [18] software package. It facilitates tasks such as simulation execution, measurement and event passing. It maintains the discrete event simulation queue, and schedules the pieces of code to be run for each specific event in the queue. This makes for an useful base upon which to build the domain specific behaviour of the simulation actors.

SimPy uses a process based approach where the simulator actor classes inherit a SimPy process class. Each actor class has a specific `activate()` method, which performs the simulation actions of the instance. It takes care of halting the execution of the method at points when the process is waiting for something else to happen, be it a specific duration to pass or an event to fire. SimPy achieves this execution halting by having the `activate()` methods return continuations, using the Python generators and the `yield` keyword [22]. This works by having the process yield a delay condition, which the SimP framework stores in the event queue, after which it moves the execution to the handler of the next event in the queue.

The basic structure of the implementation can be seen in figure 5.1. The core of the network contains five classes: Network, Route, Departure, Station and Packet. These build up the essential parts of the simulation. In addition to these classes there are classes outside the simulation package, such as the Router classes providing the routing behaviour in the `routing` package, and utility classes in the `simpy` package.

The simulation centres around the packets travelling in the network. This is noticeable from the fact that the packet class is the only one interfacing with the packets outside of the simulation. The choice to use the packet as the anchor point originates from the thesis point of view that the behaviour of the network is static, and the only variable is the behaviour of the packet. So by altering the paths of the packet, we aim to change the metrics, and thus cost of the packets.

Below are the classes of the UML class diagram explained in greater detail, along with their connections to other classes. The actual implementation of the simulation obviously contains additional boilerplate classes to handle tasks such as configuration management, simulation control and measurement output. These are not particularly interesting, and do not contribute directly to the implementation of the simulation model, nor the router algorithms, and are thus left out of this thesis.

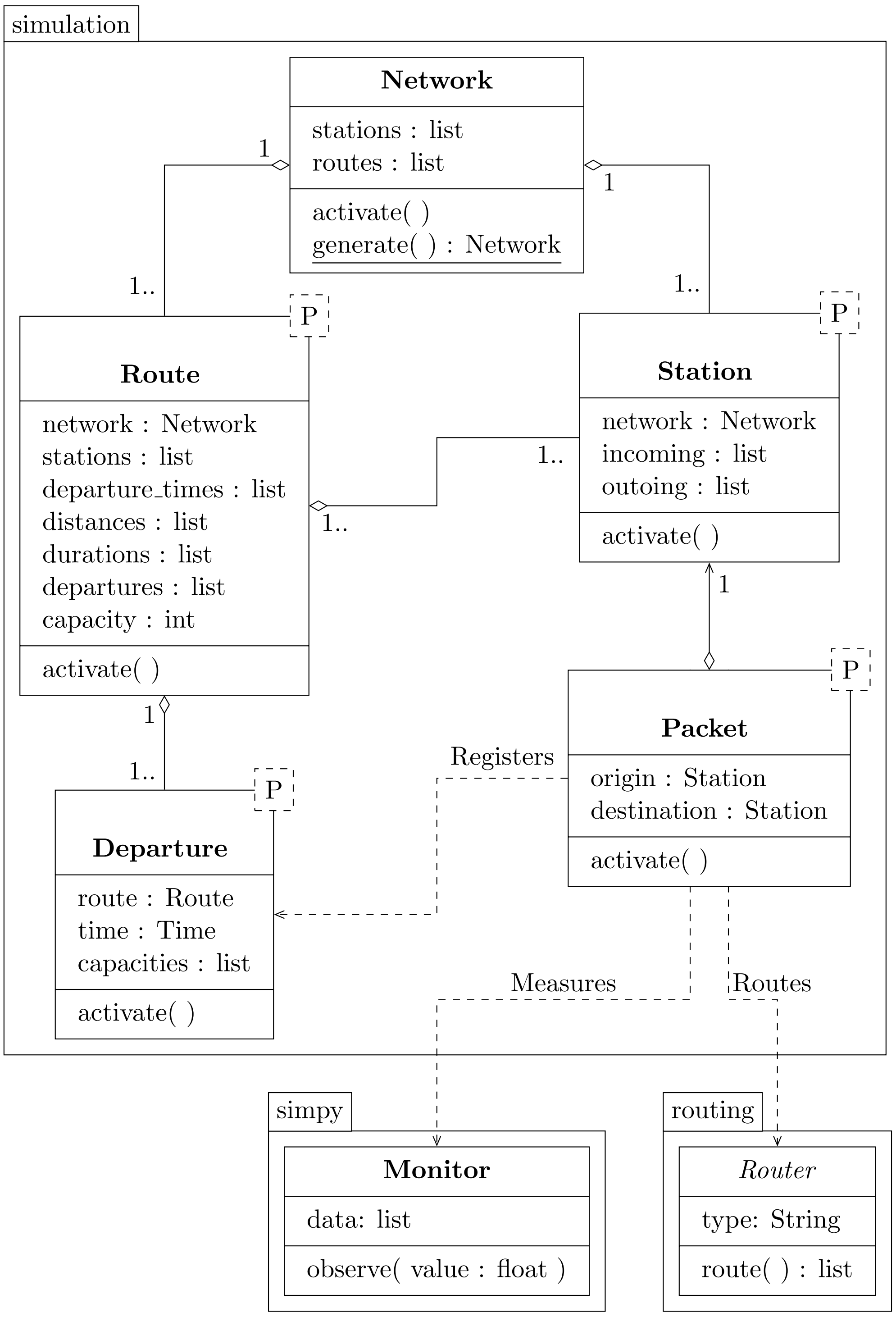


Figure 5.1: UML class chart of implementation architecture. Classes marked with a P act as SimPy processes.

Network The network class behaves as a simple container of all data stored within the network model. This includes the routes and the stations, which link further to the other classes. The network provides other classes with access to these objects, and is tasked with starting their simulation executions.

The network also provides a static method for generating random networks based on the methods in section 4.1, which is used to produce the networks for the simulation according to user-configured parameters. The generated networks are directly in-memory, and no network serialisation is available. Reproducibility of results is provided by seeds given to the random generators. The methods used to generate the random networks are described in more detail in 4.1.

Usage of real world data sets was considered, and implemented, by using publicly available data in the General Transit Feed Specification [10]. The problems with these data sets were that the networks were not connected, meaning that there are no paths from each station to each other station. This is because most public data sets are public transit networks where the traveller needs to walk by foot between some of the stations. The problem in using these networks for packet transport is that the packets are unable to move between the stations on their own, regardless of how close they would be to each other.

Route Routes are created by the network class during the network generation. They stores the transport connections between the stations as well, as the timetable information of the transport departures. Each route consist of a set of legs, along which the route travels when visiting each of the stations on the route. The legs are not classes of their own, but stored in multiple lists of stations, departures and distances. The class also keeps track of all departures of the route, and provides methods to retrieve the next departures of the route at specific stations.

Although the model actually permits unidirectional routes, the implementation enforces all routes to be bidirectional, resulting in a directed network where each also has an opposite edge. This modification was added in the implementation phase in order to ensure grater probability of achieving connected networks form the random network generation process. The connectedness of the network would otherwise become problematic at smaller sizes, or with larger amounts of simulated networks as the probability of disconnected nodes grows.

The routes also act as SimPy processes in the way that they produce and execute Departure processes, at times defined by the timetable data embedded in the route. This ensures that all departures scheduled in the network timetables are instantiated, and departing in the simulation, at the right times. All departures needed for the simulation could be instantiated at the start of the simulation, but would eat up unnecessary resources as most of the departures would stand idle. Additionally it would not make it possible to set the simulation ending criteria to be based on anything else than the simulation time. While this certainly is the most common scenario, it is not the only one.

Departure A departure is a single instance of a drive on a route. Each departure instance has a time which specifies which departure of a route it is, and at which times it will arrive at the stations. The departure instances are created by the route instance to hold departure specific information. The most important information contained in the departure are the set of packets travelling on the departure, and any further capacity reservations made for the departure. This enables the simulation to track the capacities of the departures, and ensure that no extra packets can travel on the departure after it is full.

Even the departure acts as a process in the SimPy process model. This in order for it to time the departure and arrival events of the legs on the route. The driving duration is nothing more than an event in the event queue identifying when the departure arrives at a station, so no 'active' driving is performed.

Each departure contains an list of events for each arrival and departure at a station. The packets travelling on the departure subscribe to these events, in order to be notified when to enter or leave the bus and continue execution. Upon departure from a station, the departure notifies as many packets as the departure can hold to get on the bus. And upon arriving at a station, the departure notifies all events listening to the arrival event to leave the bus.

Station Stations build the actual core of the network, as they are the origins and destinations of the packets travelling there. Each station contains a list of outgoing legs, which is used by the routing algorithms to find suitable routes for the packets to travel on.

In addition to the route connections, the stations also handle the task of producing new packets into the simulation. Each station has a probability distribution, according to which it produces new packet at random intervals. To achieve this, the Station class is also implemented as a SimPy process, waiting for random duration between packet instantiations.

Packet Packets are generated by the stations, and routed across the network by the routing algorithms. Upon instantiation each packet is assigned a random other station in the network as its destination. In order to arrive at the destination the packet *rides* departures by registering to their departure and arrival events.

The packet class is the only one that interfaces with the Router class. On instantiation, after receiving a destination station, the packet requests a path from the Router class, which provides a list of legs the packet should take in order to reach the destination. The packet then blindly follows these instructions, without making any changes nor reroutings during the journey.

It is also the class that measures most of the metrics recorded in the simulation. Upon receiving the path from the router, it stores some metrics on the path such as the distance and number of changes. Later, when arriving at the destination, the packet stores the total duration and costs of the path. It also keeps a count on the total number of packets in different states of travel during the simulation.

Router The router finds paths for all packets requesting them. It uses the origin and destination objects of the packet to find a path from the underlying network. This is an abstract class in the sense of the simulation, and actual implementations are described in detail in greater detail in section 5.2. All the implementations need to do is implement the `route()` method that takes the packet information as parameters, and returns a list of legs leading the packet to its destination.

Monitor The Monitor class stores measurements form the Packet class during the simulation. All of the measured data is stored as time-value pairs, which enables for more detailed analysis of the data post-simulation. The data is then serialised in binary format using the Python `pickle` module for later investigation. Usually the data is used for averages and variances, but there needs to be a possibility to cut warm-up and cool-down times from the data after the simulation, before averaging results for display.

5.2 Routing algorithm implementations

Each packet entering the simulation needs a plan on how to travel in the network. To retrieve this plan the packet requests from the router a path from the packets current location to its destination. The router then uses the packet station data to access the network information and search for the best path. The actual router selection is handled by the configuration boilerplate code, which is left out from this description.

To provide validity of the routing methods described in section 4.3, we implemented proof-of-concept router implementations that work with the simulation structure of the previous section. The implementations are for the abstract Router class seen in the UML class chart 5.1.

Since the three routing methods covered in this thesis are largely similar, they also share a lot of common code. Thus we implemented a common shortest path searcher class based on Dijkstra's algorithm [7], which is the de-facto shortest path solving algorithm. This class handles the network traversal and final path generation. The actual router implementations are then only tasked with defining the costs of the stations in the network.

Section 5.2.1 describes the common `route()` method of the routing class. Sections 5.2.2, 5.2.3, and 5.2.4 describe the additions to the class that define their routing behaviours for a shortest path router, a weighted cost router and a capacity reserving router.

5.2.1 Basic Dijkstra's algorithm implementation

The general Dijkstra's algorithm router was implemented as a Python class, with one public method `route(location, destination)` that returns a list of legs a packet should travel along to get from the given location, to the given destination, at the given time. The parameters originate from the packets requesting the path, and the result is used by the packet when deciding which routes to travel along.

Code listing 5.1 show Python-style pseudo-code of the Dijkstra's algorithm implementation used for the routers. It basically uses a priority queue, implemented as a heap, to keep the unvisited stations in order of calculated cost.

The `route()` method calls an abstract method `get_cost()` on line 18. This method is not present in the abstract class, but is left for the subclasses to implement. The function takes a leg, from the current station to a neighbouring station, and returns the calculated cost of this transfer. The algorithm then adds this cost of the leg and the total cost of the station, in order for it to be a new candidate for the lowest cost of the station at the end of the leg.

On line 13 the method calls a helper method `get_path()`, which returns the calculated shortest path. This is simply done by traversing the `Station.connection` links set for each traversed station on line 21. The legs of the routes become the legs of the shortest path.

```
 1 class DijkstraRouter:
 2     def route(location, destination):
 3         initialize nodes and queue
 4         queue.push(location)
 5 
 6         while unvisited is not empty:
 7             current = queue.pop_min()
 8 
 9             if current is visited:
10                 continue
11 
12             if current is destination:
13                 return get_path(destination)
14 
15             for leg in current.outgoing:
16                 neighbour = leg.destination
17                 if not neighbour.visited:
18                     cost = current.cost + get_cost(leg)
19                     if cost < neighbour.cost:
20                         neighbour.cost = cost
21                         neighbour.connection = leg
22                     queue.add(current)
23 
24             current.visited = True
```

Listing 5.1: Dijkstra's algorithm implementation pseudocode

To calculate station costs the implementation uses legs as the transitions between the stations. The legs contain all information of the distances and timetables between the stations. A leg is thus the part of a specific route that travels from one station to the next.

The Python `heapq` data structure was used as the prioroty queue. It is implemented as a binary heap, which means it is not as fast as, for example, a Fibonacci heap. But is simpler, and available in the Python standard library.

5.2.2 Shortest path router

The shortest path router implements the abstract general router class. It does this by implementing the `get_cost()` method, with code that simply returns the distance of the leg. This way the calls to this method, on line 18 of the `route()` method in listing 5.1, accumulate to be the distance of the path.

The simple pseudo code for the method can be seen in listing 5.2. Here, the leg object directly contains the needed distance, which it gets from the parent route. So all it needs to do is to return the distance.

```
class ShortestPathRouter(DijkstraRouter):
    def get_cost(leg):
        return leg.distance
```

Listing 5.2: Shortest path router implementation pseudo-code

5.2.3 Weighted cost router

While the shortest path router was able to implement the cost function concisely since it only takes into account the distance of the leg, the weighted cost router requires more complexity in order to calculate the wait and travel durations of the legs.

```
 1  class WeightedCostRouter(DijkstraRouter):
 2      def get_cost(leg):
 3          previous_leg = leg.origin.connection
 4
 5          distance = leg.distance
 6          changes = leg.has_changed_route(previous_leg)
 7          arrival_time = previous_leg.arrival_time
 8          departure_time = leg.next_departure_time(arrival_time)
 9          wait_duration = departure_time - arrival_time
10          travel_duration = leg.duration
11          duration = wait_duration + travel_duration
12
13          distance_cost = distance_weight*distance
14          changes_cost = changes_weight*changes
15          duration_cost = duration_weight*duration
16
17          return distance_cost + changes_cost + duration_cost
```

Listing 5.3: Weighted cost router implementation pseudo-code

In addition to the distance, the weighted cost function takes into account the number of changes and the duration of the path. To do this it needs to gather these metrics, and use them calculate the weighted cost. This can be seen in the pseudo-code of listing 5.3, on line 4. The distance is retrieved in the same way as in the shortest path router.

To check if the packet has changed routes between the legs the method compares the current leg with the previous leg of the departure station. The returned value of `has_changed_route()` is 1 if the route changed between the legs, and 0 if not. The implementation of this method is a simple equality comparison of the legs route objects.

To calculate the duration of the leg, the method must get both the waiting duration until the next departure, and the duration of the actual drive of the leg. The wait duration can be calculated from the difference between the arrival time of the previous leg, and the departure time of the current leg. The `next_departure_time()` method on line 8 finds the next departure of the route travelling the leg, and returns the time of departure. The actual drive duration of the leg is stored within it.

After retrieving the required metrics the weighted cost can be calculated on rows 13 to 15, using weights from user configurations, and finally returned on line 17.

5.2.4 Capacity reserving router

The capacity reserving router works much in the same way as the weighted cost router. The only difference is that the shortest path calculated by the algorithm needs to be reserved for the packet travelling the path.

```
1  class CapacityReservingRouter(WeightedCostRouter):
2      def get_path(destination):
3          path = super.get_path(destination)
4
5          for leg in path:
6              leg.reserve_capacity()
7
8          return path
```

Listing 5.4: Capacity reserving router implementation pseudo-code

To reserve the path upon finishing the traversal, the capacity reserving router overrides the `get_path()` method of the general Dijkstra router, and iterates over each leg in the path reserving capacity for a single packet on each of the legs. This can be seen in the code listing 5.4, on lines 5 and 6. For this to work, the `next_departure_time()` needs to be aware of the capacities of the departure at the leg, and provide the departure time of the next departure with free capacity.

This implementation style is obviously quite tightly coupled with the simulation implementation, with the router directly updating the data structure of the network. This means that this type of calculation would not work in a concurrent environment. For a real world implementation, the router could keep its own timetable and capacity data structures which would de-couple the router from any simulation or network state data storage. Alternatively, the router could simply use available network reservation data directly, provided by some other part of the system as a whole.

Chapter 6

Measurements and results

In order to measure the benefits of capacity reserving, compared to traditional weighted cost routing, a simulation experiment is set up using the methods and implementations from chapters 4 and 5. In section 6.1 of this chapter, we compare the effects of the three implemented routing algorithms on the efficiency of simulated networks. This gives us results on how well the usage capacity reserving information improves the throughput of the network.

In addition to this, we present the running times of the routing algorithm implementations in section 6.2. We do this in order to assert that the computational effort put into managing the extra information does not make the algorithms impractical due to increased computational costs. This does thus not actually affect the results of the thesis, but is included to strengthen the applicability of the results in practical situations.

6.1 Network performance

We construct a simulation experiment, in order to observe the actual effect of the usage of capacity reserving methods from chapter 4 in routing algorithms for transport networks. We use the network model described in chapter 3 as the level of abstraction in the simulation. The finer details of the simulation software and routing algorithm implementations used in these simulations are described in chapter 5.

Results expected from this experiment are twofold. Firstly, the validation of the behaviour of the simulation and router implementations, meaning that they exhibit behaviour similar to what can be assumed of the models and methods. This includes observing the effects of the cost functions used for the routers on the different metrics. The inclusion of the shortest path router is in order to verify this behaviour.

Secondly, and more importantly, the experiments attempts to determine the specific degree by which capacity reserving can improve the performance of the routing algorithm, and the network in general. This includes observing any possible decreases in costs, or increases in overall network capacity. This also includes determining any tradeoffs or negative effects caused by the usage of capacity reserving.

The efficiency of a routing algorithm was defined in section 4.4.2 as it's ability to increase the network throughput; or in other words, it's ability to transport multiple simultaneous packets to their destinations. We take this as the main point of measurement in this evaluation. This means that the more packets the network can process without saturating, the better the router performs. Saturation can be measured by the sustainability of the network, which is measured as the ratio of packet entering to packets exiting the network during a measured time. We thus perform the experiments as a series of simulations with incrementally increasing packet loads, in order to see how many packet the different routers are able to efficiently route in the networks.

Additionally we need to monitor the cost of the packets, along with other metrics, in order to see that the router does not start to sacrifice too much in other regards in order to keep the throughput high. For example, a router that would drive unnecessary long routes in order to increase the efficiency only slightly, would in many cases not be a satisfactory result.

The predicted results from the simulation, based on the planned behaviour of the routing algorithms, is that the shortest path router should provide the distinctively shortest paths, while suffering from a significantly poorer overall cost. On the other hand, the weighted cost and capacity reserving routers should share the simulation results up until some certain packet load, at which the results will start to deviate as the departure capacities start to fill up in certain areas of the network.

For the total network performance, it is predictable that the shortest path router and the weighted cost router would approximately share the same results, while the capacity reserving router will be able to better handle an increased load. At which point this happens, and to what degree, is the main point of interest in the measurement experiment. There is also a very real possibility that the network capacity will be adversely affected by the capacity reserving router, in contrast to the other ones, in situations where the load is significantly higher than the capacity of the network. In these situations the *extra* legs taken by the capacity reserving router might use too much of the, at this point, scarce transporting resources.

6.1.1 Simulation setup

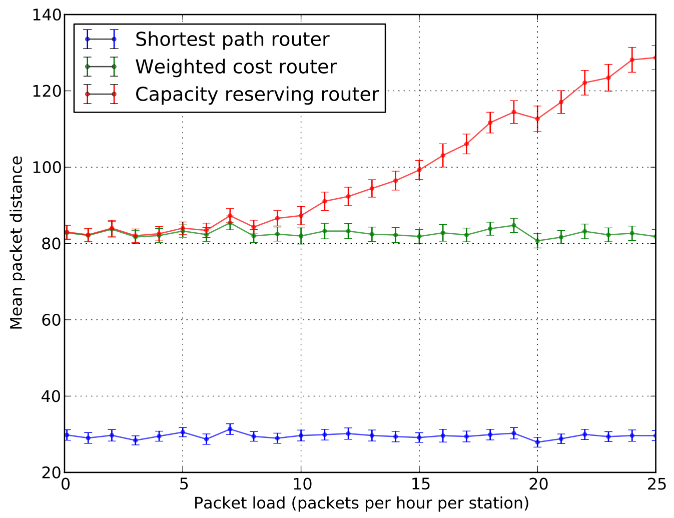
The experiment set-up consists of sets of simulations of random transport networks, using the different routing algorithms. The simulation methods used are described in more detail in section 4.2 on packet routing simulation. The simulated networks contain 20 stations, each station having an average of 5 transport lines leaving from the station, each with an average of 3 stops along the way. Each line has an average of 6 departures a day, and carries an average of 15 packets per departure. Each line stops for one minute at each stations, and each packet takes an average of 45 minutes to transload to another line at a station.

The values used represent a group of networks with certain attributes. They, along with the resulting measurements, do thus not represent all possible networks, but instead gives an indication to some possible set of networks and results. The values were specifically chosen to enable the different routers to present substantial improvements in the network efficiency. This means providing them with high enough levels of connectedness and variation to enable multiple different possible paths. This also means that the amount of packets transported needs to be near the total capacity of the network, in order to give any value for the capacity load information.

Although these simulations only measure the different routing algorithms ability to handle packet loads, it is important to note that these are not the only factors that affect the performance of the network. The size and connectedness of the network as well as the capacity and frequency of the departures both affect the base performance of the network. These parameters also affect the behaviours of the routing algorithms.

As the networks are randomly generated, each simulated network has different characteristics, such as how the stations happen to be grouped, and how well they happen to be connected. This means that there is no such thing as an average network, and the same applies for the generated packets. That is why simulations need to be run for multiple networks and packet traces, in order to observe the average behaviours of the routing algorithms.

Each network was simulated for 240 hours of simulation time, with packet loads ranging form 0.1 to 25 average packets per hour per station. A warm-up time of 48 hours, and a cool-down time of 120 hours were used, meaning that any packets sent between the 48th hour and the 120th hour were not taken into account in the visualised results. The large cool-down times are required for all of the packets sent during the measurement period to reach their destinations, and provide proper measurements for their complete journey. Each configuration was run 100 times, with different random networks and packet traces.



(a) Data plot

Load	S	W	C
0.1	29.8	82.8	83.0
1.0	29.0	82.1	82.3
2.0	29.7	83.7	84.0
3.0	28.4	81.7	82.0
4.0	29.5	82.1	82.6
5.0	30.5	83.3	84.0
6.0	28.7	82.3	83.4
7.0	31.3	85.4	87.3
8.0	29.4	82.0	84.3
9.0	29.0	82.5	86.6
10.0	29.7	82.0	87.3
11.0	29.9	83.3	91.1
12.0	30.2	83.3	92.3
13.0	29.7	82.4	94.4
14.0	29.4	82.2	96.5
15.0	29.1	81.9	99.3
16.0	29.6	82.8	103.1
17.0	29.4	82.2	106.1
18.0	29.9	83.9	111.7
19.0	30.3	84.7	114.4
20.0	27.9	80.7	112.7
21.0	28.8	81.6	117.0
22.0	30.0	83.2	122.1
23.0	29.4	82.3	123.4
24.0	29.7	82.7	128.1
25.0	29.6	81.8	128.7

(b) Data table

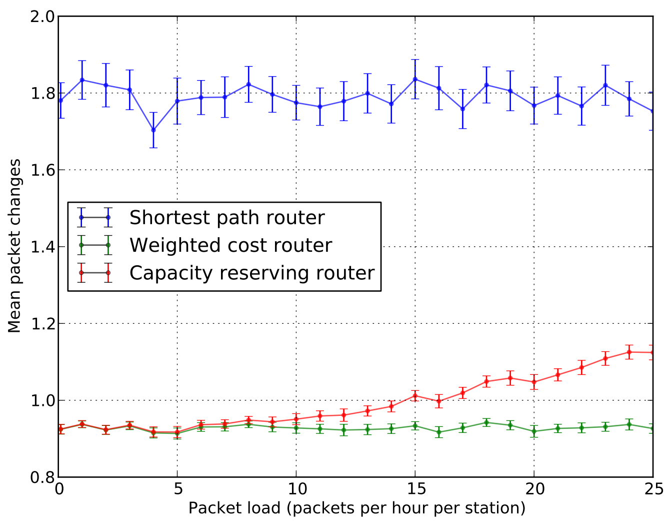
Figure 6.1: Mean packet distances for increasing packet loads. Error bars are the 95% confidence limits of the SEM.

6.1.2 Results

The results from these simulations are presented as averages over the multiple simulation runs. For all measurements except the sustainability of the network, the average was calculated as the arithmetic mean, and the errors as the 0.95 confidence limits from the standard error of the mean (SEM) of the averaged values. None of the measurements can contain negative values, meaning that the distribution isn't strictly normal, but they are close enough though to provide usable results. For the sustainability of the network the median was used as the average, and the median absolute deviation (MAD) as a measure for error. The median was chosen for the sustainability since it does not follow a standard deviation, and needs a robust measure for meaningful results. The average averaged results of these simulation runs are shown in figures 6.1, 6.2, 6.3, 6.4 and 6.5.

Distance Figure 6.1 shows the mean travelled packet distances of the simulated networks, using the three routers. We can directly see that the shortest path router provides the shortest routes, with an mean path distance of 58 km, compared to 98km of the weighted cost and capacity reserving routers.

The mean distances are static for the shortest path and weighted cost routers over the increasing packet load, due to the routers always choosing



(a) Data plot

Load	S	W	C
0.1	1.78	0.92	0.93
1.0	1.83	0.94	0.94
2.0	1.82	0.92	0.92
3.0	1.81	0.93	0.94
4.0	1.70	0.91	0.92
5.0	1.78	0.91	0.92
6.0	1.79	0.93	0.94
7.0	1.79	0.93	0.94
8.0	1.82	0.94	0.95
9.0	1.80	0.93	0.94
10.0	1.77	0.93	0.95
11.0	1.76	0.93	0.96
12.0	1.78	0.92	0.96
13.0	1.80	0.92	0.97
14.0	1.77	0.93	0.98
15.0	1.84	0.93	1.01
16.0	1.81	0.92	1.00
17.0	1.76	0.93	1.02
18.0	1.82	0.94	1.05
19.0	1.81	0.94	1.06
20.0	1.77	0.92	1.05
21.0	1.79	0.93	1.07
22.0	1.77	0.93	1.09
23.0	1.82	0.93	1.11
24.0	1.78	0.94	1.13
25.0	1.75	0.93	1.12

(b) Data table

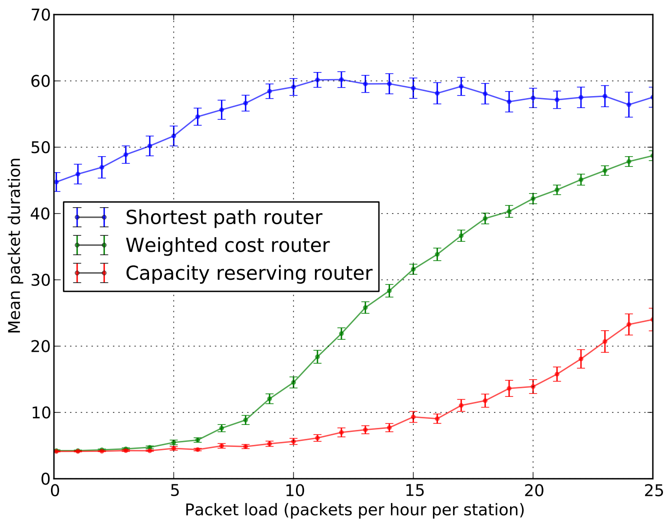
Figure 6.2: Mean number of packet changes for increasing packet loads. Error bars are the 95% confidence limits of the SEM.

the same routes regardless of the network load. The capacity reserving router on the other hand sees significant increase in distance after the 10 packets per hour per station mark, as it routes packets around congestion points at higher network loads.

There are noticeable fluctuation in the average results of the measurements for different network loads. The bumps in the plots seem to occur at the same points for all routers. This indicates that the source of the fluctuations is the unpredictable and varying nature of the networks used for the simulations. The amplitude of the differences, even after averaging over 100 networks, just show how much variation the network set contains. This wiggle could be reduced with additional simulation runs and means over larger data sets, but the trend is in any case evident.

Changes Similarly to the distance results in figure 6.1, the mean number of changes in figure 6.2 show very static behaviour for the shortest path and weighted cost routers. The capacity reserving router also exhibits the same behaviour of re-routing the packages around bottlenecks when a certain capacity is met. From the results we can see that the divergence point seems to be at five packets per hour per station, after which both the amount of changes, and the distances, start to increase for the capacity reserving router.

It is interesting to see that the shortest path router operates remarkably



(a) Data plot

Load	S	W	C
0.1	44.8	4.2	4.1
1.0	45.9	4.2	4.1
2.0	47.0	4.4	4.2
3.0	48.9	4.5	4.2
4.0	50.2	4.7	4.2
5.0	51.7	5.5	4.6
6.0	54.6	5.8	4.4
7.0	55.6	7.6	5.0
8.0	56.6	8.9	4.9
9.0	58.4	12.1	5.3
10.0	59.1	14.5	5.6
11.0	60.2	18.4	6.1
12.0	60.2	21.9	7.0
13.0	59.5	25.8	7.4
14.0	59.6	28.3	7.7
15.0	58.9	31.6	9.3
16.0	58.1	33.8	9.1
17.0	59.2	36.7	11.1
18.0	58.1	39.2	11.8
19.0	56.9	40.3	13.6
20.0	57.4	42.2	13.9
21.0	57.2	43.6	15.8
22.0	57.5	45.1	18.1
23.0	57.7	46.5	20.7
24.0	56.4	47.8	23.3
25.0	57.5	48.7	24.0

(b) Data table

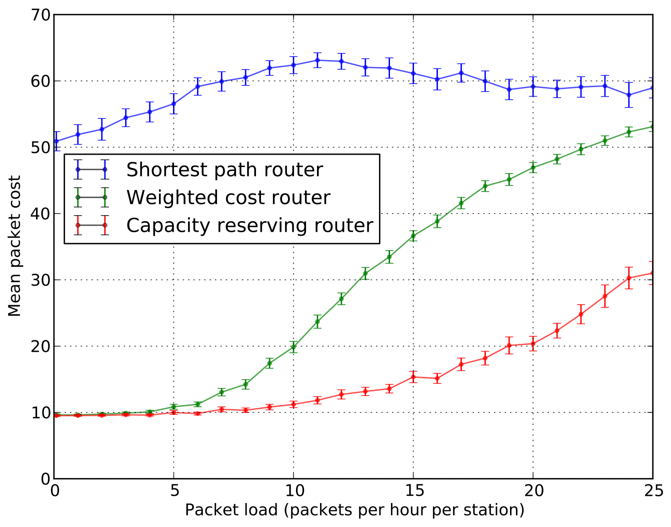
Figure 6.3: Mean packet durations for increasing packet loads. Error bars are the 95% confidence limits of the SEM.

well in the number of changes with an mean of under two changes per packet. Of course this is directly a property of the underlying network, dependant on how the routes are placed, and what distances they have. On a different set of networks, with different parameters, this would probably have looked quite different.

Although the cost function for the capacity reserving router puts a lot of weight on the number of changes, the capacity reserving router still needs to route the packets over more legs to satisfy the time constraints. Nonetheless, the rate of divergence is clearly smaller compared to the divergence for the packet distances.

Duration While the packet load did not affect the static routers in the previous measurements, the duration is clearly affected by the load for all routers, as can be seen in figure 6.3. For small loads, we see that the shortest path router takes the longest time as its cost function disregards all notion of duration. The weighted cost and capacity reserving routers share a significantly smaller duration for small loads, due to the duration relevant weight in the cost function.

The durations for all routers increase along with the network load, as was to be expected. The difference is in the degree they are increasing. As was already noted in the previous figures the capacity reserving router diverges



(a) Data plot

Load	S	W	C
0.1	50.9	9.6	9.5
1.0	51.9	9.6	9.5
2.0	52.7	9.8	9.6
3.0	54.4	9.9	9.6
4.0	55.3	10.1	9.6
5.0	56.5	10.9	10.0
6.0	59.1	11.2	9.8
7.0	59.9	13.1	10.5
8.0	60.5	14.2	10.3
9.0	61.9	17.4	10.8
10.0	62.4	19.9	11.2
11.0	63.1	23.7	11.8
12.0	63.0	27.1	12.7
13.0	62.0	31.0	13.2
14.0	61.9	33.5	13.6
15.0	61.1	36.6	15.3
16.0	60.2	38.8	15.1
17.0	61.2	41.6	17.3
18.0	60.0	44.1	18.2
19.0	58.7	45.1	20.1
20.0	59.1	46.9	20.4
21.0	58.8	48.2	22.3
22.0	59.1	49.7	24.8
23.0	59.2	51.0	27.5
24.0	57.9	52.3	30.3
25.0	58.9	53.1	31.0

(b) Data table

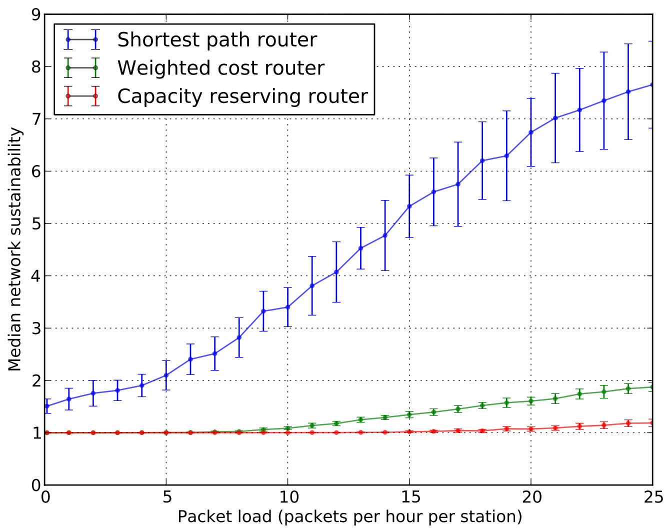
Figure 6.4: Mean packet costs for increasing packet loads. Error bars are the 95% confidence limits of the SEM.

from the performance of the weighted cost router at packet load levels of five packets per station per hour, and above.

It is interesting to note how the gap between the capacity reserving router and the weighted cost router decreases as the network packet load grows even higher. This is most likely due to the fact that the capacity reserving router is, in a sense, wasting unnecessary departure capacity when trying to reroute past congestion points. This works well for smaller congestions, but when the complete network is filled up, it only makes matters worse.

Cost Figure 6.4 shows the mean costs of the simulations, which the weighted cost and capacity reserving routers aim to minimise. These values are largely dominated by the durations of the packets, and the plots follow the same shape. Nonetheless, we see that the capacity reserving router is able to decrease the individual packet cost by half under higher packet loads.

The results are slightly biased against the shortest path router, since the cost function used for calculating the cost in the figure is the same cost function as used in optimising the other routers. This does not affect the relation between the weighted cost and capacity reserving routers, which is the main point of importance in this measurement.



(a) Data plot

Load	S	W	C
0.1	1.51	1.00	1.00
1.0	1.64	1.00	1.00
2.0	1.75	1.00	1.00
3.0	1.81	1.00	1.00
4.0	1.90	1.00	1.00
5.0	2.10	1.00	1.00
6.0	2.40	1.01	1.00
7.0	2.51	1.02	1.00
8.0	2.82	1.02	1.00
9.0	3.32	1.06	1.00
10.0	3.40	1.09	1.00
11.0	3.81	1.14	1.00
12.0	4.07	1.18	1.00
13.0	4.53	1.25	1.01
14.0	4.77	1.29	1.01
15.0	5.33	1.35	1.02
16.0	5.60	1.39	1.03
17.0	5.75	1.45	1.04
18.0	6.20	1.52	1.04
19.0	6.29	1.57	1.08
20.0	6.74	1.61	1.07
21.0	7.02	1.65	1.09
22.0	7.17	1.74	1.12
23.0	7.35	1.78	1.14
24.0	7.52	1.84	1.18
25.0	7.65	1.87	1.19

(b) Data table

Figure 6.5: Median of the packet sustainability for increasing packet loads. The error bars are the MAD.

Sustainability The results presented above mostly relayed information about the individual packets travelling in the network, and only hinted at what actually occurred on an network-wide scale. To get a closer look at the performance of the network, we also measured the sustainability of the network. Sustainability of a network is the rate at which the amount of packages increase in the network. A sustainability value of 1 means that the network can handle the packet load nicely, while anything above means that the buffers will be filling up. The sustainability was discussed in more detail in section 4.4.2. The results of the sustainability measurements are seen in figure 6.5.

We can directly see that the shortest path router is unable to keep up with the packet load at any level. This is due to the fact that the shortest legs of the system will be clogged up instantly, since almost all packets travelling larger distances will be routed through some central shortest legs. The weighted cost and capacity reserving routers on the other hand handle things better. The cost function spreads the load of the packets on a wider set of routes that minimise the amount of changes and duration, without sending all packets to the same departure.

The ability of the capacity reserving router to additionally spread the load around congested routes, enables it to be sustainable at higher packet loads than the simple weighted cost router. This can be seen in the data table 6.5b, where the capacity reserving router stays sustainable up until a packet load of 12, compared to the weighted costs routers value of 5. This provides an increase of 100% to the total network capacity.

Even at higher loads, the sustainability is better for the capacity reserving although it is not level. This eases the network in occasional spikes, such as holidays or other special events, where the need surpasses the capacity momentarily. In these cases, buffers such as storage warehouses will not fill up as rapidly, and the packets will arrive in a more timely manner.

6.2 Algorithm running time

Since finding the cheapest path is not a trivial problem, and since the networks can grow large and routes complex, we cannot assume that the computational requirements of the algorithms are trivial. Thus, it is important to be aware of the overhead brought on from he more complex methods. Even the best methods can become infeasible, if they are too complex to perform in real world applications within reasonable time limits.

All algorithms compared in this thesis are based on Dijkstra's algorithm, as described in section 5.2. It has an worst case running time bounded by $O((n + e)\log e)$ using a binary heap for the priority queue, where n is the number of nodes and e is the number of edges in the network. Dijkstra's algorithm builds the whole shortest path tree in that time, but since we only need the shortest path to a specific node, we end the computation once this node is reached in the breadth-first search.

From the knowledge that the average number of edges in the shortest path of a scale-free network is approximately logarithmic to the size of the network [1], we can estimate an average running time for the routing algorithms in our model. Let us say that the length of the shortest path is $l = \log n$, and that the branching factor of the network is b. Then each edge added to the shortest path increases the number of nodes to consider by b. From this we can deduce that the total number of nodes to consider in the shortest path search will be $\log^b n$, which means that the running time is expected to grow polylogarithmically with the size of the network.

The additions to the capacity reserving algorithm in section 5.2.4 increase the computation needed for each node by a constant amount, since the capacities of the path departures need to be checked. This could decrease performance slightly, but should still stay in the same time complexity.

It is also worth noting that while network size is a big factor in the routing time, the connectedness of the network also affects this. The more connections the packets have to choose from, the larger the search space grows. In the running analysis, the amount of connections affects the branching factor of the breadth-first search tree. For simplicity the effect of connectedness on the routing time was not taken into account in these measurements, since the main point is only to assert the general performance of the algorithms.

6.2.1 Simulation setup

To verify that the algorithms presented here actually are computationally efficient enough for practical usage in larger networks, we measured the wall-clock time of the routing algorithm execution time in simulated situations. Network sizes of the simulated networks were increased linearly from 10 to 1 000 stations, with each station having an average of 2 routes, which in order having average lengths of 3. This produces a branching factor of 6 in the shortest path search. The algorithm execution was first warmed up by routing 100 packets, after which 1 000 packets were routed and timed. For each network size and routing algorithm, 50 simulations were run with different random networks and random packets.

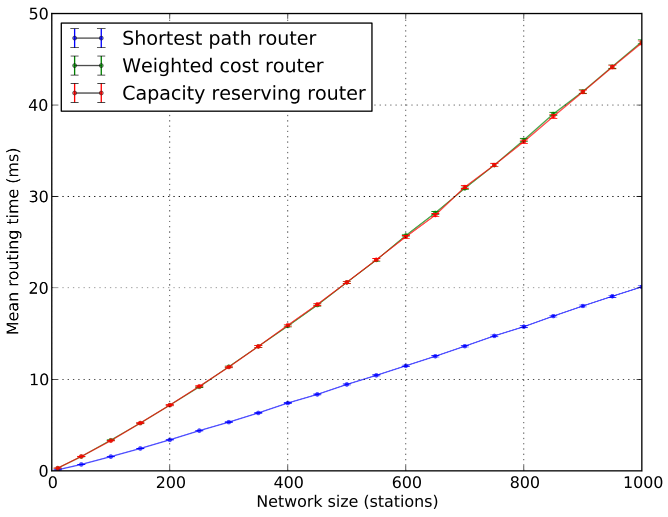
The simulations were limited to 1 000 stations, as running the simulation running times became infeasible long at this point. In a real world applications, the routing algorithms would obviously be completely detached from any simulation structure, enabling more efficient data structures and faster performance in larger networks.

Hardware used for the performance measurements was a system with a Intel Xeon E3-1230 processor running at 3.20GHz and 8 GB of memory. The system ran Ubuntu 12.04 LTS and CPython, the reference Python implementation, version 2.7.3.

6.2.2 Results

In figure 6.6 we can see the mean running times of the algorithms, for increasing networks sizes. The values seem to follow a linear pattern, which means the algorithms should scale well with increasing network sizes. Even at 1 000 stations in the network, we have a running time of under 50 milliseconds, which is still very manageable for real-time routing.

The weighted cost and capacity reserving routers have distinctively different running times, compared to the shortest path router. This means that adding the time factor to the cost model roughly doubles the amount of work needed to be done at each step while traversing the network. This is mainly due to having to check the timetable data for the next available departure and fetch its arrival times.



(a) Data plot

Size	S	W	C
10	0.1	0.3	0.3
50	0.7	1.6	1.6
100	1.6	3.4	3.3
150	2.4	5.2	5.2
200	3.4	7.2	7.2
250	4.4	9.2	9.2
300	5.3	11.4	11.3
350	6.3	13.6	13.6
400	7.4	15.8	15.9
450	8.4	18.1	18.2
500	9.4	20.6	20.6
550	10.4	23.0	23.1
600	11.5	25.7	25.6
650	12.5	28.2	28.0
700	13.6	30.9	31.0
750	14.8	33.4	33.4
800	15.8	36.2	36.0
850	16.9	39.0	38.7
900	18.0	41.5	41.4
950	19.1	44.2	44.1
1000	20.1	46.9	46.8

(b) Data table

Figure 6.6: Mean running times for the three algorithms in differently sized networks. Error bars are the 95% confidence limits of the SEM.

It is important to remember, that the algorithm implementations used in these performance measurements were prototypical in their nature, and implemented in Python using simple unoptimised object-oriented approaches. Running the routing algorithms independently from the simulation, while using a more optimised structure and algorithms, would increase the performance even further. Additionally, more specialised and sophisticated routing algorithms, compared to Dijkstra's algorithm, could be used as a base for the implementation.

Chapter 7

Discussion

The previous chapter presented simulations and results comparing the different routing algorithms. The results contained promising values, and in this chapter we take a look at what implications those results might have for a transport service provider. We also discuss the rigidity of the methods used to achieve the results. Furthermore, we take a look at how the methods could be incorporated into real world usage scenarios.

7.1 Implications of the results

The results from section 6.1 showed that capacity reserving can decrease the individual packet cost by up to 50%, under heavy load, in certain network scenarios. Using the specific cost distribution model in the simulations, the lowered cost was largely due to the decreased packet travel time, at the expense of increased distanced and slightly increased amount of changes. As the algorithm is able to adapt to the increased load under these circumstances, according to the cost function at hand, means that similar results should be seen in other circumstances as well.

A decreased individual packet cost means direct savings for the a network provider. In the cost model used, the provider would pay slightly more for fuel due to increased packet mileage, and slightly higher costs for transloading packets at stations. But these would be earned back from the customer satisfaction of the decreased packet duration. Of course, this is directly dependant on the cost model used, and with a different cost splits the results would be different. But similar results should be seen in total cost reduction as long as the durations of the packets are taken into account.

Additionally, the results showed that the network was able to withstand up to 100% higher packet loads, using the capacity reserving router, before the amount of packets entering the system outgrew the throughput of the network. This is a direct consequence of the algorithms ability to distribute the load of the network across multiple varying paths. This is not bound to the cost model used, as long as the model takes in anyway into account the travel time of the packets.

An increased network capacity, by being able to handle a larger load, means that existing networks will be able to sustain larger utilisation growth by using capacity reserving, in comparison to using traditional weighted cost methods. This enables a transport service provider to make large long term savings since costly network equipment investments can be pushed back, and any investments put into increased network capacity have a larger effect on utilisation.

There are only a few situations where he capacity reserving routing could provide results worse than the traditional weighted cost routing. The differences in the routing algorithms only surface when the capacity of a departure is full. At this point all packets travelling on that path will be rerouted to more cost-effective alternatives. This decreases the cost of the packets on this specific path. On the other hand, the rerouted packages might fill up other departures, and thus hinder the travel of packages on other routes. This can actually be seen in the results at maximal loads, where the additional routes travelled by the packet waste the available resources of the network.

Although the reserving router showed increased performance in the specific network set used in the results, it does not mean that the savings will be the same for all network structures. The network topology and the cost model of the packets greatly affects the usefulness of the capacity reserving methods. As an extreme example acyclic networks, such as hub-and-spoke networks, will not see the benefit from reserving routing. This is since each packet has only one possible path from its origin to its destination in these networks. Or, if the cost model disregards any duration aspects, then any rerouting of the packets would only increase the costs due to longer distances and additional changes.

The algorithms are developed specifically for diverse and vast networks with varying packet routes, meaning that there is little room for micromanagement and per-case optimisation. So in addition to the connectedness of the network, the degree in which the network and the routes are optimised to the load of the packets, greatly affects the effectiveness of the capacity reserving. For example, a transport network with very static consignments is able to specialise its connections, routes and vehicles to distinctly cater to the specific consignments shipped in the network. In these cases the reserving router will not have enough head room to reroute packets any more effectively.

In addition to using capacity reserving as a tool to improve transport networks, it could also be used to improve more general routing problems. Some examples were already given of use in load aware communications network routing algorithms as stated earlier in section 2.2.3, but it could be brought even further.

7.2 Discussion on methodology

The simulations run in chapter 4 are based on a simple model and randomly generated networks using randomly generated loads. Although the models are formed to match real world networks as closely as possible, the results are only as accurate as the model is. This means that the results the above implications are based upon, do not necessarily transfer to the real world scenarios as they are.

The largest caveats to take into account, when considering the methods used to obtain the results, are two-fold: not using enough real world data, and not providing any verification of the implementations. The random network models make some assumptions, and are rather simple in some regards. Without any measured comparison with real world networks, they might not always provide accurate representations of real world scenarios. The simulation implementations can unfortunately not be made publicly available, and were only verified by the feasibility of their measurement results.

All in all, this means that the results are indicative but not definitive. For greater actual data on real world performance, some kind of trials would need to be done in the environment of final deployment. This could, for example, mean taking the deployment network and some test load data and see the difference in performance with different routing methods. In short, the simulations would need to be run in real world scenarios, and no guarantee is provided that they will work as-is.

7.3 Usage in applications

The capacity reserving routing algorithm needs a way to follow the network state. Integration with a system does not require much more than knowledge of the network structure and timetables. The routing algorithm can, if necessary, simulate the network by itself to keep track of the capacity loads in the network. Optionally, if the real world data is stored elsewhere, then the routing algorithm could directly integrate with any provided real world information that might be available. This would be the preferred way to do read the network state data, since any deviations from the routing suggestions would be taken into account.

The packet routing can be done directly at first contact with the packet, so that the planned path could be sent along any other data on the packet. If the packet misses a departure or travels off the path, then re-routings could be requested from the system as needed. This means that parcel labels, counters and hand-terminals could always show the suggested path of the packet, giving the manual sorting much needed instructions on where to send the packet.

The results presented in this thesis make a lot of assumptions on the reliability of the transports in the network. The only precaution against delayed transport is the transloading time, which sets a minimum duration between planned arrival at station and the next departure. Any real world application of the algorithms should take into account misplaced packets, significantly delayed departures or missing departures, and handle them accordingly.

Chapter 8

Conclusions

In this thesis we set out to improve transport network efficiency by using packet routing algorithms that take into account the network capacity load at each point in time. By using this additional piece of information, we hoped to increase the performance of the network, especially under high packet load.

After identifying the structure and model of network the packet routing occurs in, we produced the methods needed for executing the model, and constructed a simulation environment upon which to perform measurements. We introduced the capacity reserving router, which was able to provide a significant improvement in network efficiency compared to traditional routing methods.

The work done in this thesis can be well applied in specific areas areas of transport and communication, where there is a specific need to be able to balance peak load over multiple connections. This could include parcel transport in postal services, or information communication networks such as mesh packet routing. Additionally, the results might be suitable for use in the broader field of graph theory, where they might contribute towards a more general approach for load balancing of specific types of flow networks.

Although the capacity reserving router is in a completed state as it is, there is still room for future improvements. As an example, each packet has multiple optional paths, each with their own costs, and the network has a total cost which is the sum of the packet costs. So instead of minimising the cost for single packet entering the system, we could minimise the total cost of the network. In practice, this would mean that new packets entering the system would be able to alter the paths of packets already travelling in the system, if such a change would result in a reduced total cost. This would clearly increase the routing algorithm search space significantly, but could nonetheless provide practical in smaller instances combined with specialised caching and heuristics.

Alternatively, the network data itself could be used to identify packet load patterns in order to predict future resource needs and route packages based on this knowledge. All required information would already be available to the routing system from the incoming routing requests. This would not require rerouting of packets already in transit, but could still be prove useful in predicting and avoiding congestions.

The real-time load data maintained by the capacity reserving router could also be used for other applications. Possible congestion points, and high payload departures, can be identified beforehand from the made reservations, giving loaders and drivers additional time to prepare for any exceptional situations. For example, higher packet amounts may take longer to load or handle, meaning more time or resources would be needed to perform the task.

Additions such as these could be included as separate extensions, or as a combination, depending on how well they perform in which contexts. Nonetheless, they could provide valuable additions to the underlying framework introduced by this thesis.

Bibliography

[1] Albert, R., and Barabási, A. Statistical mechanics of complex networks. *Reviews of Modern Physics 74* (2002), 47–97.

[2] Ancillotti, E., Bruno, R., Conti, M., and Pinizzotto, A. Load-aware routing in mesh networks: Models, algorithms and experimentation. *Computer Communications 34*, 8 (2011), 948–961.

[3] Barabási, A., and Albert, R. Emergence of scaling in random networks. *Science 286*, 5439 (1999), 509–512.

[4] Bellman, R. On a routing problem. *Quarterly of Applied Mathematics 16* (1958), 87–90.

[5] Berliant, M., and Watanabe, H. A scale-free transportation network explains the city-size distribution. In *RCEF 2012: Cities, Open Economies, and Public Policy* (2011).

[6] Cooke, K. L., and Halsey, E. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications 14*, 3 (1966), 493–498.

[7] Dijkstra, E. A note on two problems in connexion with graphs. *Numerische mathematik 1* (1959), 269–271.

[8] Figueiredo, L., Jesus, I., Machado, J., Ferreira, J., and Martins de Carvalho, J. Towards the development of intelligent transportation systems. In *2001 IEEE Intelligent Transportation Systems Proceedings* (2001), IEEE, pp. 1206–1211.

[9] Fredman, M. L., and Tarjan, R. E. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM 34*, 3 (1987), 596–615.

[10] Google. *General Transit Feed Specification Reference.* `https://developers.google.com/transit/gtfs/reference`.

[11] Helsinki Region Transport. Reittiopas. `http://www.reittiopas.fi/en`.

[12] Hesse, M., and Rodrigue, J. The transport geography of logistics and freight distribution. *Journal of transport geography 12*, 3 (2004), 171–184.

[13] Janic, M. Modelling the full costs of an intermodal and road freight transport network. *Transportation Research Part D: Transport and Environment 12*, 1 (2007), 33–44.

[14] Kajosaari, R., Langius, E., and Holmström, J. State of the art in tracking based business. In *The 13th International Conference on Concurrent Enterprising* (2007), ICE Conference.

[15] Klock, R., Owens, D., Schwartz, H., and Plencner, R. Integrated intermodal passenger transportation system. Tech. rep., NASA, 2012.

[16] Law, A. *Simulation modeling and analysis*, fourth ed. McGraw-Hill, 2007.

[17] Lee, S., and Gerla, M. Dynamic load-aware routing in ad hoc networks. In *The IEEE International Conference on Communications* (2001), vol. 10, IEEE, pp. 3206–3210.

[18] Muller, K., and Vignaux, T. SimPy: Simulating systems in Python. *ONLamp.com Python Devcenter* (2003).

[19] Ricci, A. Pricing of intermodal transport. Lessons learned from RECORDIT. *European Journal of Transport and Infrastructure Research 3*, 4 (2003), 351–370.

[20] Rodrigue, J., Comtois, C., and Slack, B. *The Geography of Transport Systems*, second ed. Taylor & Francis, 2006.

[21] Santos, J. L. Real-world applications of shortest path algorithms. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, vol. 74 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.* AMS, 2009, pp. 1–19.

[22] SCHEMENAUER, N., PETERS, T., AND HETLAND, M. L. PEP 255 – Simple generators in Python. `http://www.python.org/dev/peps/pep-0255/`, 2001.

[23] SKIENA, S. *The Algorithm Design Manual*, second ed. Springer, 2008.

[24] SOLAKIVI, T., OJALA, L., TÖYLI, J., ET AL. *Logistiikkaselvitys 2010*. Liikenne- ja viestintäministeriö, 2010.

[25] STOCK, J., AND LAMBERT, D. *Strategic logistics management*, fourth ed. McGraw-Hill, 2001.

[26] VAN ROSSUM, G., ET AL. Python programming language, 1994.

[27] ZANJIRANI FARAHANI, R., REZAPOUR, S., AND KARDAR, L. *Logistics Operations and Management: Concepts and Models*. Elsevier, 2011.