Imperial College of Science, Technology and Medicine
Department of Computing

Congestion Avoidance in Overlay Networks Through Multipath Routing

Victor Faion

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of the Imperial College London and
the Diploma of Imperial College, October 2013

I herewith certify that all material in this dissertation which is not my own work has been properly acknowledged.

Abstract

Overlay networks relying on traditional multicast routing approaches use only a single path between a sender and a receiver. This path is selected based on latency, with the goal of achieving fast delivery. Content is routed through links with low latency, ignoring slower links of the network which remain unused. With the increasing size of content on the Internet, this leads to congestion, messages are dropped and have to be retransmitted.

A multicast multipath congestion-avoidance routing scheme which uses multiple bottleneck-disjoint paths between senders and receivers was developed, as was a linear programming model of the network to distribute messages intelligently across these paths according to two goals: minimum network usage and load-balancing. The former aims to use as few links as possible to perform routing, while the latter spreads messages across as many links as possible, evenly distributing the traffic. Another technique, called message splitting, was also used. This allows nodes to send a single copy of a message with multiple receivers, which will then be duplicated by a node closer to the receivers and sent along separate paths only when required.

The model considers all of the messages in the network and is a global optimisation. Nevertheless, it can be solved quickly for large networks and workloads, with the cost of routing remaining almost entirely the cost of finding multiple paths between senders and receivers. The Gurobi linear programming solver was used to find solutions to the model. This routing approach was implemented in the NS-3 network simulator. The work is presented as a messaging middleware scheme, which can be applied to any overlay messaging network.

Acknowledgements

I would like to thank the following people:

- My supervisor, Alexander Wolf, for guidance and inspiration.
- My second supervisor, Peter Pietzuch, for insightful ideas.
- Antonio Carzaniga and Paolo Costa for helpful discussions and advice.
- Wolfram Wiesemann for invaluable discussions on modelling and also for patiently helping me with CPLEX and Gurobi.
- The Computing Support Group for letting me use their servers to run my experiments.
- My friends for allowing me to stay motivated while having fun.
- My girlfriend for being very supportive and patient, especially during the later stages of my PhD.
- My family for their unconditional support throughout my PhD.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

The problem addressed in this work is how to avoid congestion faced in overlay networks. These types of networks usually use a single path between a sender and a receiver. Traditional routing approaches in overlay networks select paths based on latency, trying to achieve fast delivery. The chosen path between a sender and a receiver is the path with the least latency. This greedy approach to path selection can cause a build up of packets at nodes connected to low-latency links as many packets will be routed through them. These nodes may not have the processing power to deal with so many packets and will end up dropping them. This results in possibly multiple retransmissions until enough resources are available to process the packets.

When packets are routed through the same node at the same time a collision occurs. As network activity increases, the number of collisions increase, causing latency as well as reliability problems. Applications using TCP will have their transmission rates limited once many collisions are detected. However, typical applications which use UDP will not be rate-limited and packet loss will not be detected. As messages are sent into the network without restriction, not only will the application experience packet loss, but also the whole network will be affected.

Users which are using different applications (even those using TCP), will also experience poor performance caused by the congestion.

As IP-layer multicast is done over UDP, applications which rely on this communication pattern need to be careful not to overload the network. Some applications use an application layer multicast which is implemented over TCP, but this typically does not provide the potential for reducing congestion that IP-layer multicast provides. When applications which use IP-layer multicast send a message addressed to multiple receivers, it is sent only once, and intermediate nodes propagate the message to the appropriate nodes. However, in application layer multicast, this optimisation is not performed, requiring the sender to send the message individually to each receiver, as a series of unicasts.

Another potential problem with this type of multicast is that because it relies on TCP, receivers will acknowledge the receipt of messages or signal their loss. This type of traffic can cause further collisions, and even if it all reaches the sender, the sender can be overwhelmed by the processing required, being left unable to send new messages.

Therefore, when timely delivery is important, the typical application layer multicast approach is not used. For example, the popular chat application, Skype, relies on UDP for sending media traffic [BFS09]. In general, real-time multimedia applications, such as video conferencing generate a large amount of multicast traffic, which has a high priority, yet it has to compete evenly with traffic with lower priority. Other types of popular multimedia applications, such as streaming video and music, also generate large amounts of traffic, but this is typically not multicast traffic. As real-time traffic requires timely delivery, typically TCP cannot be used [Per05]. Therefore, applications which generate this type of traffic do not have the congestion control provided by TCP. As the use of real-time applications with no congestion control increases, there is concern among the Internet standards community about the disruption that can potentially occur [Per05].

Another risk leading to dropped packets is that a fast link may not necessarily have the correspondingly large capacity required to handle the large amounts of traffic which will be routed

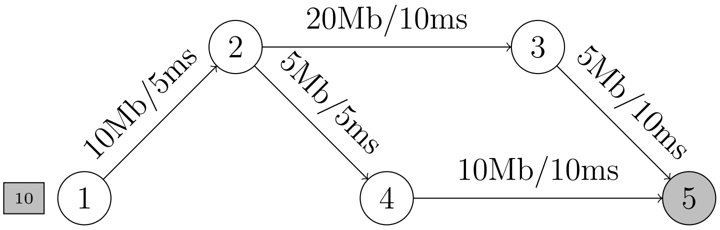

Figure 1.1: Messages going from node 1 to node 5 at a rate of 10Mbits per second.

through it using the traditional path selection technique used in overlay networks, where paths are selected based on latency without considering capacity. This means that messages are not delivered in the quickest way possible, which was the reason for picking the paths in this way to begin with. In this work, a 'message' refers to an overlay message, which can be made up of one or more packets. Similarly, a 'flow' is made up of one or more messages. When talking about the underlying network, usually packets are referred to, while the term 'message' is used when referring to the application layer.

An illustration of this is presented in Figure 1.1. The links are labelled with their capacities in MBits and their latencies in milliseconds. Suppose that node 1 wants to send messages to node 5 at a rate of ten Mbits per second. In a typical overlay network, this traffic would be routed via the path $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$ because this is the fastest path by latency. However, the edge $2 \rightarrow 4$ has only five MBits of capacity and therefore packets will be dropped and retransmitted.

Congestion control schemes have been proposed to deal with this problem, usually by slowing down the rate of transmission. They measure available bandwidth or simply use dropped packets as a signalling mechanism that congestion has occurred [LBS$^+$08, ZDA06, PB03]. To the best of my knowledge there is no accurate and inexpensive way to measure available bandwidth in a network and relying on crude congestion detection mechanisms using dropped packets e.g., [EJLW01], is undesirable. A different routing approach in which packets are routed around congested nodes instead of dropped and retransmitted is proposed. They go through nodes which have the most resources available.

This is combined with another way of alleviating congestion: using multiple paths to route messages to their destinations. Using a set of paths between a sender and a receiver, Senders are allowed to split the flow of messages they want to send across the set of paths. This technique is called multipath routing and it has been shown to increase throughput [BO07, WWK$^+$07]. Having multiple paths provides more network resources for sending a flow of messages and is less likely to result in congestion. This should allow messages to be delivered more quickly than in the traditional routing approach as there is no rate limiting and there is less strain placed on the network as retransmission of dropped packets is avoided. In the work, this technique is referred to as path splitting.

Referring back to Figure 1.1, path splitting can be taken advantage of by splitting the traffic from node one across two paths in order to avoid dropping packets. In this case the traffic can be split in half across the paths $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$ and $1 \rightarrow 2 \rightarrow 3 \rightarrow 5$. By sending only five MBits per second across the link $2 \rightarrow 4$, overloading it and dropping packets is avoided. This also allows node three to be used to process half of the traffic.

1.2 Messaging Middleware

With the spread of the Internet, enterprise systems are becoming increasingly complex and heterogeneous, frequently spanning across different geographical areas [JBB12]. These systems produce large amounts of data and have many diverse sets of producers and consumers of information. This information needs to be aggregated, handled uniformly, and distributed according to the increasing demands of applications. This is a crucial part of today's enterprise systems.

Message-oriented middleware allows applications to communicate without knowledge of each other: messages are simply exchanged between producers and consumers, or publishers and subscribers, with the messaging middleware serving simply as a bus between applications [ASD09]. These systems provide a communication layer between the applications, abstracting away details related to operating systems and different networks. This allows developers to develop applications across multiple operating systems and network protocols without being concerned with specific details related to integration [SS12]. The diversity which needs to be handled by messaging middleware systems brings huge challenges with respect to guaranteeing a uniform level of quality-of-service across different applications running in different networks.

Modern business-critical systems depend on robust messaging middleware. If messages are lost, delivered late or not processed properly, there is real, economic impact [Kra09]. As a single message can represent a large transaction, reliability is crucial and missing messages is not an option. This type of systems are heavily used in commercial as well as scientific data centres for high-performance computing [SYD13]. Sensor systems also rely on messaging middleware for message delivery. This kind of data is also very time sensitive as the validity of data is time dependent and messages delays can have catastrophic consequences [YKK$^+$09].

Messaging middleware systems frequently follow a publish-subscribe model, providing many-to-many communication. This decoupling of publishers and subscribers allows for greater flexibility in terms of interoperability between applications, as well as path reconfiguration in changing network conditions [KKY$^+$10]. This is also matches the increasing decoupling of geographically distributed enterprise systems. The distance between systems means that messages have to travel over long paths with increased risk of congestion and packet loss due to the increased number of nodes and links which are traversed [YKK$^+$09].

There are various different messaging middleware systems, suitable for different environments, but a common feature among them is that they are implemented at the application layer and they rely on the layers underneath to handle routing and network congestion. As congestion is not proactively avoided, other measures are taken by the users of these types of systems, such as replication to provide backup in case of dropped packets and failures. Despite the increased risk of delays occuring, as compared to a local network, little concern is given to attacking this problem directly. Delays due to congestion are expected and measures are taken to mitigate their impact rather than eliminate their cause. These are costs that could be avoided if the root of the problem is considered.

Messaging middleware is becoming important not just in the business environment, but on a global scale as well. With the increasing amount of sensors in the world, the data which needs to be managed by messaging middleware will increase. Smart homes, smart grids, various types of environmental and disaster monitoring are all areas which are generating large-scale producers and consumers of information and messaging middleware is crucial in managing this data and meeting the demands of publishers and subscribers.

1.3 State of the Art

To the best of my knowledge there is no single protocol or model which combines all the features of the protocol. In this section, work from various fields related to this work is described. This is discussed in more detail in the next chapter.

1.3.1 Messaging Middleware

The most closely related area is messaging middleware systems. These are frequently based on the publish-subscribe communication model and are used as a way for heterogeneous enterprise applications to communicate without worrying about different protocols, formats and other integration issues. Some messaging middlewares provide guarantees for different levels of quality-of-service [WBW11, ASD09, O'H07, KKY+10, YKK+09]. Some of them take into account congestion [WBW11, ASD09, SM12, KKY+10, YKK+09], but usually only latency is considered. Some messaging middlewares use a service-oriented architecture [GE09, HZZ+09, JSK+11]. This is a design pattern in which independent software components are used to provide functionality, for example a messaging layer, to applications as a service. Some messaging middleware systems use multipath routing [WBW11, KKY+10, YKK+09], but not in the same way that it is used in CAMPR (Congestion Avoidance through MultiPath Routing), described in more detail in Sections 1.3 and 1.4. Instead of splitting traffic across multiple paths, they simply replicate the traffic across multiple paths, as a way to improve resiliency.

Another type of messaging middleware is content-based networking. These type of systems are relevant because they also provide a publish-subscribe communication pattern. As the demand for content becomes greater, content-delivery networks and peer-to-peer systems have been applying content-based functionality in their overlay networks. These services also rely on the routing and congestion control schemes provided by the underlying network architecture, but recently approaches which consider routing and how to deal with congestion in the context of a content-based network have emerged [CGMP13].

Much of the work done in this field is based on the work done by Alexander Wolf and Antonio Carzaniga [CW01, CRW06, CRW09]. There have been many variations of routing messages in a network based on their content. The main theme among these protocols, implemented at the application layer, is to match the content of a message to the subscriptions of users in the network. A notable example is content-centric networking, being worked on by researchers from the Palo Alto Research Center [JMSGLA07]. Some of these protocols are discussed in the next chapter.

1.3.2 Multipath Routing

Another field related to this work is multipath routing. This refers to the technique where a sender will divide the messages they are sending along multiple paths going to a receiver. Usually these protocols are not multicast protocols and simply deal with sender-receiver pairs. In this work, this multipath technique is used and combined with an overlay multicast. Sometimes multipath routing refers to duplicating the messages across multiple paths for more reliability, but this is not done in this work.

There are various ways that multipath routing protocols select the multiple paths used for routing. Sometimes the paths are fully disjoint [MFC08, IN02, BO07], not disjoint [WWK$^+$07, IUY06, XJT09, MP09] and sometimes they are bottleneck-disjoint [NZ01]. In this approach, bottleneck-disjoint paths are paths which may share common links (i.e., they may not be disjoint), but they may not share a bottleneck link. The bottleneck link of a path is defined as the link with the least capacity in the path. The bottleneck-disjoint path selection method was found to be the most effective and is the one used in this work.

There are also different techniques used to rank the available paths between senders and receivers. Distributed [WWK$^+$07, XJT09, MP09] and centralised [MFC08, IUY06, IN02, BO07] approaches to multipath routing have been proposed. The distributed approaches do not assume global knowledge of the network, however the solutions of these approaches are not as accurate as a global solution. The approach taken in CAMPR is a centralised one. Some of the multipath routing approaches treat the problem of distributing traffic across multiple paths as an optimisation problem [IUY06, IN02, BO07, XJT09]. They represent the problem as a mathematical model. This is the approach taken in this work.

1.3.3 Traffic Engineering

Another closely related field is traffic engineering. The work in this field looks at how to make the most of available network resources, and the techniques in this field are used by ISPs. One of the main goals of the work in this field is to find a way to load-balance traffic and minimise congestion. This is also one of the goals of this work. Traffic engineering solutions also have distributed approaches and the load-balancing problem is treated as an optimisation problem [AMG05, EJLW01, HBCR07, KKDC05].

The field of bandwidth reservation was also looked at, a technique used to manage network resources in order to provide a quality of service (QoS) to applications such that user demands are met. This technique can be used in conjunction with traffic engineering. It is frequently used in mobile ad hoc networks, but sometimes in stationary networks as well, which is the context in which it is discussed [ABIS06, SKY11]. Multipath routing is also sometimes used in bandwidth reservation approaches [SWW$^+$04, ER04]. The goal is to efficiently use network resources in order to allow the maximum amount of traffic.

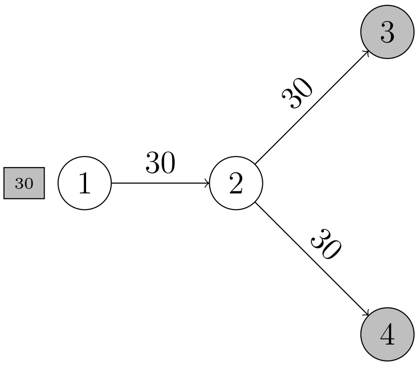

Figure 1.2: Message splitting occurs at node two.

The multipath routing, traffic engineering and bandwidth reservation approaches do not take into account message splitting – when a node can send a single copy of a message subscribed to by multiple receivers, which will be duplicated by a node closer to the receivers and sent along separate paths only when needed.

An example of message splitting is depicted in Figure 1.2. In this case node one wants to send thirty messages meant to be received by nodes three and four. In a traditional overlay network, node one would send the thirty messages twice, but using message splitting, node one sends the thirty messages only once. At node two, message splitting occurs and the thirty messages are sent out twice, going to nodes three and four. By splitting messages close to their receivers, network resources are saved. Message splitting is formally defined in Section 3.3.

How to do multipath routing taking into account all of the senders in the network which are sending messages to multiple receivers is the problem solved. This means considering how the network resources are shared between all the senders. This requires global knowledge of the available resources in the network. This is a fair assumption as some of the centralised multipath routing approaches presented make similar assumptions [MFC08, IUY06, IN02, BO07].

Different levels of knowledge of the network allow for different things. For example, UDP requires no knowledge of the network load or capacity. Once a packet is sent using UDP, there are no guarantees that it will be delivered and no delivery acknowledgements. The sender does not know if the receiver has received the data or if resending is required. The receiver may not be able to receive packets at the sending rate and may be flooded by packets, having to drop the ones it cannot handle. In a sense, the receiver is assumed to be infinitely fast as the sender has no knowledge of the receiver's speed.

TCP provides reliable delivery, the endpoints are more aware of the load and receivers are not assumed to be infinitely fast. Senders attempt to prevent overflow caused by sending packets too quickly through flow control. Messages are acknowledged and receivers send requests for dropped packets to be resent. Senders have indirect knowledge of congestion in the network, which they can gauge through the delays of acknowledgements from receivers. When the sender detects congestion it will slow down its sending rate to match the receiving rate. When congestion occurs, the only way to alleviate it is to throttle the sending rate. At the receiver, TCP passes packets up to the application layer in order. This may cause packets which have been received out of order to be retransmitted, if buffer space runs out. In general, there is more communication between senders and receivers about the network state as compared to UDP.

Multipath TCP has as much knowledge of the network as TCP, but it is also aware of multiple paths provided by the routing protocol, rather than just a single path as in TCP. Multipath TCP acts as glue between multiple TCP connections joined to act as one: if congestion is detected on a path, the sending rate can be reduced on this path and increased on another, less-congested path (unacknowledged data is sent along another path). There is indirect knowledge of congestion along the multiple paths. Multipath TCP also provides in-order delivery and this makes the protocol more complicated than TCP, as well as requiring more control traffic. This is still a greedy approach and all of the traffic in the network is not considered.

In CAMPR, there is knowledge of all of the traffic in the network and this allows for more optimisation than with the other protocols. A major difference is that in addition to being a multipath protocol, CAMPR is also a multicast protocol. When determining the paths along which to route messages, all of the network traffic is considered and paths which have enough capacity are selected. For simplicity, a centralised approach was chosen. This is a first step towards a more distributed approach, which would require more control traffic to distribute the information which is assumed to be globally available. In a real protocol, there could be a node which collects network information (link capacities and message loads), generates and solves a model representing this, and sends out the relevant parts of this solution to the corresponding nodes.

CAMPR is a congestion avoidance scheme at the overlay level. Overlay networks are easy to deploy as the underlying network does not need to be changed. They also allow for the realisation of services which are not or cannot be supported at lower levels of the network. IP level multicast exists however it requires support from the network and it is not widely deployed. IP level multicast does not provide congestion avoidance.

At the lower levels, UDP can support multicast, but it does not do anything to avoid or alleviate congestion. TCP has only local path knowledge and does not avoid congestion, but slows down traffic once congestion has occured. Multipath TCP uses multiple paths, which are given to it by the routing protocol, however, these paths are not chosen because they meet traffic demands (i.e., the paths may not have enough capacity and packets will be dropped). Neither TCP nor multipath TCP support multicast. The overlay layer is appealing because there is a global view of the network traffic.

1.4 Contributions

The main contributions are summarised:

- A model generator for a multipath, multicast, message splitting routing protocol
- A path selection technique which can be used in any routing protocol
- A general model which can be applied to any overlay messaging network
- A simulation of CAMPR which uses the solutions from the models to perform routing

The main contribution is a model generator used to generate a model of a multicast protocol which allows for message splitting and path splitting. Hereafter, the approach is referred to as CAMPR (Congestion Avoidance through MultiPath Routing). To the best of my knowledge this is the first protocol to combine these features.

The model generator takes a network topology and a message workload and generates a mathematical model which represents the given state of the network including the available resources in the network. The available resources are represented as constraints which must be met assuming that all of the messages are to be delivered in the given topology. Bottleneck-disjoint paths from the senders to the receivers are computed and used to build up the delivery graphs (the set of paths from a sender to the receivers) for each message flow. This information is also used in the model. To the best of my knowledge this is the first such model that considers message splitting and path splitting.

The model is used to find a routing approach which results in a message flow distribution which is meant to avoid dropping packets given the resources in the network. The model is solved with a linear programming solver and the solution tells us how to route the given workload of messages. This means knowing how to distribute message flows across delivery trees such that all traffic fits within the capacity of the network. The solution can be used to construct the forwarding tables in a routing protocol and message traffic can be split accordingly.

This novel routing scheme is presented in the context of messaging middleware however, it can be applied to general overlay networks. The bottleneck-disjoint path selection technique in particular is completely independent from the nature of the protocol. The model could be used for any overlay network with path splitting and message splitting.

A simulation of CAMPR was also implemented in a modified version of NS-3 which allows users to use topologies from BRITE.[1] In this simulation, the paths calculated were used to generate the models and routing was performed according to the solutions to these models.

[1] http://www.nsnam.org/wiki/index.php/BRITE_integration_with_ns-3

1.5 Preview

A small preview of the rest of this thesis is given. Starting with the next chapter, related work in the area of messaging middleware (Section 2.2) is discussed. Work done on protocols in the field of content-based networking (Section 2.3) is also looked at. Several different techniques used to deal with congestion (Section 2.4) are discussed. First, congestion control schemes applied in overlay networks are discussed, including a technique called backpressure – a hop-by-hop feedback mechanism for signalling short-term congestion. These schemes mostly rely on measuring available bandwidth in links and using rate limiting to slow down packet transmission as a way to alleviate congestion. This should be avoided as there is no reliable and inexpensive way to measure available bandwidth. Slowing down sending rates should also be avoided.

Backpressure is one technique used to deal with congestion. When congestion is detected nodes send feedback back to the senders and the senders slow down their sending rates. This approach adds extra control traffic, requires nodes to maintain extra state and does not always improve network performance. Congestion control schemes do not prevent dropped packets. In fact a problem with current congestion control schemes occurs when packets are dropped due to congestion and a node signals multiple senders for retransmission. This will usually result in dropped packets again as they will be retransmitted simultaneously. Expensive packet retransmissions should be avoided.

One way of increasing the available capacity for a message flow is to split it across multiple paths. For example two paths can be calculated between a sender-receiver pair and the sender can split their message flow evenly across the two paths. This technique is called multipath routing and the principles of this type of routing are applied to this work, splitting message flows across multiple delivery graphs. Multipath routing has been shown to increase throughput [BO07, WWK$^+$07]. In fact multipath routing can also be combined with a congestion control scheme, but a different approach is taken.

Many different schemes have been proposed which make different decisions regarding the distribution of traffic across multiple paths. Another aspect of multipath routing is how to select the multiple paths. The approach taken with regards to path selection is to use bottleneck-disjoint paths. From experimental results, this proved to be the most effective path-selection technique. Work done in this area is looked at in the context of routing in the application layer with standard TCP running on each overlay path (Section 2.5).

Then the field of traffic engineering (Section 2.6) is looked at. This field has similar goals to the goals of this work: reducing or avoiding congestion and load-balancing traffic. These techniques are commonly used by ISPs in order to make the most of the network bandwidth they have available for their users. Specifically, traffic engineering approaches where multipath routing was used are looked at. The approaches in this field are similar to the approach of this work,in that they treat this as an optimisation problem.

Some techniques used in the field of bandwidth reservation (Section 2.7) are discussed. Work in this area aims to efficiently allocate network resources such that the demands of applications in the network are met and a certain quality of service is provided. A few approaches where multipath routing is used to achieve this are looked at. The goals of methods in this field are similar to ours: to make the most of network resources in order to support a greater traffic load.

The context of this work is a store-and-forward network in which multiple senders (publishers) send messages to multiple receivers (subscribers). Each message may have many or no subscribers. Messages are to be received as quickly as possible, and as efficiently as possible, without overloading the network. Some assumptions are made in order to make this a tractable problem, such as complete knowledge of the topology, which is common in non-distributed schemes. Complete knowledge of the messages in the network is also assumed: which nodes are publishers and which are subscribers. This assumption is common in publish-subscribe networks.

The approach taken is to represent the state of the network, publishers, subscribers and the messages as a mathematical model which can be solved using optimisation software. The solution of the model describes how to route messages. In this work, this means how much of each message flow to send down which delivery graph; how to split the message flows. From this forwarding tables can be constructed. The generation of the model involves calculating (possibly multiple) paths between all the senders and receivers.

An example using a small topology and message workload are provided in order to illustrate how the approach works in Section 3.2. This is used to give the reader an intuitive understanding of some of the features of the protocol, such as message splitting and path splitting without getting bogged down in details. The running example has three parts of increasing complexity: a one sender and one receiver scenario, a one sender and multiple receivers scenario and lastly, a multiple senders and multiple receivers scenario. The concepts presented informally in the example are then formally defined in Section 3.3.

The goals of this approach are discussed in Section 3.4. The overall goals are maximum flow and minimum latency. This can be described as routing as many messages as possible from senders to receivers as quickly as possible. This is to be done on a global scale, for all the senders and receivers in the network.

These two goals often have different solutions. Achieving maximum flow is attempted through two different subgoals. One is to minimise network usage, sending messages using as few links as possible. The other subgoal is load-balancing. In the load-balancing approach, messages are spread evenly across as many links as possible. The minimum latency goal implies using fast links to route messages. This is treated as a secondary goal, using paths with enough capacity for a particular flow of messages over a faster path.

The path selection and path ranking techniques used are discussed in Section 3.5. Path ranking refers to selecting k paths for a sender-receiver pair from all available paths. At first, an early attempt at ranking paths according to their capacities is discussed. Then, the latency ranking approach which does not take capacity into account is looked at. Even though capacity is not taken into account when ranking the paths, the selected paths are bottleneck-disjoint with respect to capacity. Finally, another attempted approach which was again using latency ranking, but subsequent paths are accepted only if they have a greater capacity is looked at. The bottleneck-disjoint approach was the most successful.

The mathematical models which are generated using the state of the network and the selected paths are defined in Section 3.6. There are models for two goals: minimum network usage and load-balancing. At first the optimal models are discussed so that the reader knows the ideal goals to be achieved. These models are intractable so they could not be compared directly against the suboptimal models. After this, the suboptimal models are discussed, which are the ones actually used. A model which was used to count the number of undelivered messages using a traditional overlay routing approach which has no path splitting is also discussed. This model was used to show that congestion occurs when using a single path between senders and receivers.

The evaluation framework used is discussed. This includes the evaluation process (Section 4.2), the various parameters explored and the environment used (Section 4.3). Results from the two suboptimal models discussed previously (minimum usage and load-balancing goals) are presented, showing how many paths it takes to find a feasible routing solution given a message workload and topology. Graphs from these models are included, showing the time taken to find these solutions.

Graphs using the model which is meant to represent traditional overlay routing are presented in Section 4.4. These graphs show the number of undelivered messages with different message workloads. As a large parameter space was explored, only a subset of the graphs is presented and analysed.

The implementation of a simulation of CAMPR is described, showing that multipath routing can be performed according to the solutions of the models using path splitting and message splitting in Section 4.5. The models do not take into account time, so this was done to verify that their solutions can be used to actually build a routing protocol and distribute message workloads appropriately.

The routing process is explained, including how message splitting and path splitting are implemented, as well as how the solutions of the models were used to construct the forwarding table in Section 5.2. Simulations were run with various parameters for both the minimum usage and the load-balancing goals, discussed in Section 5.3. This simulation can also be used to make a prototype of CAMPR with minimal changes, however, this is left as future work.

Finally, areas in which this work can possibly be applied are discussed in Section 6.2. As the core routing process is not directly tied to any particular network or protocol, so it can be applied to other fields as well. The limitations and scalability of CAMPR are discussed in (Section 6.3) and possible routes for future work are discussed in (Section 6.4).

Chapter 2

Background

2.1 Introduction

In this chapter, related work in the fields of messaging middleware, content-based networks, congestion control, multipath routing, traffic engineering and bandwidth reservation are discussed, as research carried out in these fields aims to solve similar problems. This work explores some problems faced in these fields: congestion, dropped packets, network utilisation and load-balancing. As CAMPR combines ideas from these fields into a multicast multipath congestion-avoidance routing scheme with message splitting, some unique challenges are also faced.

2.2 Messaging Middleware

The popular messaging middleware Tibco [Bro11] provides a multicast message delivery service, but this relies on IP layer multicast, i.e., the network must be multicast-enabled. Tibco also provides traditional message delivery, where delivered messages are acknowledged and if not delivered, the sender attempts to resend them. However, in the multicast message delivery service, there is no message receipt acknowledgment, so a network failure or dropped message due to congestion can cause many undelivered messages, which will go undetected.

In [GE09] a messaging middleware that is meant to satisfy the needs of real-time enterprise applications is presented. This relies on the publish-subscribe paradigm. There is an event queue for publishing applications and an event queue for subscribing applications. The queues represent activities controlled by the scheduler. There are two types of schedulers, Weighted Fair Share (WFS) and Earliest Deadline First (EDF). The WFS scheduler allows an administrator to set different rates to the different queues, and offers more flexibility than EDF, but it is non-trivial to find the settings which will lead to a desired performance level. The EDF scheduler attempts to minimise the maximum lateness and can find this without configuration. This is implemented in Java and experiments measuring end-to-end latency and throughput are performed. The focus of the work is on low latency, but perhaps this could be improved if congestion avoidance and the underlying paths chosen for sending messages were considered.

A QoS-aware publish-subscribe message-oriented middleware is presented in [WBW11]. This system uses mirror brokers to provide redundancy in case of faults in the network. This middleware also performs overlay multipath routing. Rather than distribute traffic across the multiple paths, they are used for redundancy. Similar to CAMPR, traffic is allocated in such a way as to avoid overloading brokers. Source-routing is used and end-to-end delay measurements are performed and used as the basis for the overlay routes. The goal is to maximise delivery rate. This system supports criticality levels and a maximum delay for each topic.

The authors state that service degradation typically occurs when traffic increases and brokers' processing rates are exceeded. If congestion causes a path to no longer satisfy QoS requirements, a new path is calculated. In this sense, the protocol is a reactive one, waiting for packets to be dropped. The authors do not discuss this, but typically in approaches which use only delay to determine paths, some oscillation between the choice of paths occurs. The authors mention that in the future they will create a model which also takes into account the brokers' capacities.

In [ASD09], a self-healing messaging middleware is discussed. This publish-subscribe messaging middleware uses a hierarchy of topics to organise messages. The system can detect faults, which include network faults, congestion, security vulnerabilities and others, and recover from them automatically by using a new instance of the system. The system also learns from these faults, constantly improving, adapting and tuning itself so that it can prevent or better react to future faults. This is an interesting idea and distinguishes this work from other middleware messaging systems.

In some sense it is a reactive system, relying on faults to improve itself, however, it also proactively tries to minimise future faults. The system provides QoS, with some of the QoS metrics being latency and loss rate. Redundant publishers are used to increase resiliency. The system probes the network to measure loss rate and packet delay and when an anomaly is detected, the probing rate is increased. When loss is detected, relay brokers are used between the nodes experiencing the loss. The authors state that through these probes a view of the entire network is maintained and that this messaging middleware scales up to tens of nodes

In [ETR+13], messaging middleware's performance is evaluated in virtualised environments, such as the cloud computing service provided by Amazon. This paper cites work published in 2010 which also evaluates Amazon's cloud service, observing unstable network throughput. In this paper, the authors set up a testbed with similar hardware to Amazon's in order to measure the performance difference of the messaging middleware due to the cloud network. The messaging middleware compared uses TCP. The results indicate current messaging middleware systems need to become more efficient to overcome the overhead incurred by cloud networks. Cloud service providers are deploying high-speed networks to try to solve the current latency issues, but perhaps messaging middleware also needs to be adapted to this context.

A middleware system to deal with congestion faced in cluster computing is discussed in [SM12]. This is a centralised approach and is implemented in Java. This system relies on code mobility to minimise the amount of communication required. This is an interesting idea, and various problem sizes have been evaluated, but the system is only tested on clusters of up to 11 nodes. As the number of nodes increase, the communication overhead increases.

A global event notification middleware called Herald is presented in [CJT01]. In this system events are data provided by publishers for a set of subscribers at some point in time. This data is not interpreted by Herald, nor are filtering or complex subscriptions supported. In-order delivery is also not guaranteed. This work focuses on scalability issues and aims to be resilient like the Internet, functioning even when parts of the network are down. This is done through a federated approach. The design approach is having a simple base layer which is very scalable and multiple higher-level layers which provide additional functionality.

A goal of this work is to be scalable in the number of subscribers and publishers, subscriptions, delivery rates and federated domains. Another goal is automatic adaptation to load patterns and changing resource availability. The authors note that overlay networks are an effective way to distribute content to many interested parties and therefore dynamically generated overlay network is used between Herald nodes for even distribution from publishers to subscribers. Multicast-style overlay communication is used to prevent the same data being transmitted over a network link multiple times. Load-balancing is done reactively. When too much traffic occurs at a machine in the system, some or all of this traffic is shifted to another machine. The authors point out that the system must be able to support rapid changes in load due to sudden popularity increase.

The publish-subscribe middleware, Proxy, used in CERN is described in [SYD13]. Proxy replaced a middleware system which was not able to keep up with the increasing demands of the data intensive applications used at CERN. This low-latency system is written in C++ and uses CORBA (Common Object Request Broker Architecture)[1]. Each subscriber has a separate message queue so that slow consumers do not block other clients. The system is made of up of 26 servers and has been deployed since 2009. The API from the previous middleware was not allowed to change and so Proxy wastes about 50% of CPU time performing unneeded serialisation and deserialisation.

The Advanced Message Queuing Protocol (AMQP) is described in [O'H07]. This is an open

[1] `http://www.corba.org/`

topic-based publish-subscribe messaging protocol and implementations from different providers are interoperable. This came from the need for an open standard for messaging middleware systems after the author was continuously faced with solving the same problems. It is common for large organisations such as banks to develop their own messaging middleware systems, but as there was no standard, these systems come and go. The AMQP middleware can work on different network transports including TCP and UDP and messages of unlimited size are supported. Different message priorities and QoS levels are also supported. Network flow control is done by receivers who 'issue credit' to links to avoid receiving too many messages [ASB10].

There are several implementations available, and one particular implementation of this protocol called Apache Qpid is described in [Kra09]. This supports direct addressing, publish-subscribe topic-based addressing and XML-based addressing. The article mentions that although in theory, implementations from different vendors should be interoperable, in practice they support different versions of the standard and have different options, so it is best to use brokers and clients from the same vendor.

The Apache Qpid system is benchmarked in [SMN$^+$08] for various numbers of publishers, subscribers, message sizes and addressing types (called Exchange types) as well as a stock market simulation. The authors mention that the centralised design of AMQP is a significant bottleneck to the scalability of the system. As the number of consumers increase, the achievable message rate decreases for all Exchange types and the CPU utilisation at brokers is a bottleneck. The authors advocate the need for distributed brokers to alleviate the performance bottlenecks.

A cloud-based messaging middleware called Cloudqueue is presented in [SS12]. The authors claim that existing middleware systems are insufficient to support applications in the Internet of Things. This messaging middleware is based on the Apache Hadoop project[2]. This project provides a simple way to process large data sets across clusters of computers. The authors claim that Cloudqueue is more scalable than traditional messaging middleware systems and provides higher throughput. It also provides cross-data-center messaging and handles node

[2] `http://hadoop.apache.org/`

failures. Cloudqueue relies on the distributed filesystem provided by Hadoop to provide a message persistance layer. The authors compare their work to ActiveMQ, an implementation of AMQP. In the results shown, Cloudqueue provides better throughput and reacts better to node failures.

A service-oriented messaging middleware is described in [HZZ+09]. Service-oriented architecture is an architecture design pattern in which independent software components are used to provide functionality to applications as a service. This messaging middleware provides a messaging service layer between heterogeneous enterprise applications, guaranteeing reliable messaging. It is divided into three parts: a message processing handler, which is responsible for messaging sending and receiving, a common service, which handles routing, and a privacy layer, for encryption and description of messages. This system supports different priorities for messages. A mobile e-commerce use case is presented. A user uses their mobile phone to pay for an electronic document and a third-party company is responsible for providing reliable fund transfers. The authors claim that in this case, the mobile operator, document provider and payment handling company can all use their messaging middleware as it is service-oriented, and that a traditional messaging middleware system would not be suitable as it does not provide cross-enterprise application integration.

Another service-oriented messaging middleware is presented in [JSK+11]. This system is called the National e-Governance Service Delivery Gateway (NSDG) and is used for messaging between government departments in India. Services have been integrated at the national, state and local government levels. Using this middleware has given the government great flexibility in integrating its heterogeneous services. Before the introduction of this middleware, there was a tight coupling of the systems involved and many different communication protocols were being used. Among louser coupling, other benefits include authentication between services, nationwide interoperability, prevention of data loss and transaction management with logging and auditing. The middleware only reads the part of messages that it needs for routing and authentication, while the message bodies are read by the appropriate service. The authors claim that this middleware can scale up to 1000 messages per second with an average payload of 60 KB.

A method for verifying publish-subscribe messaging middleware is discussed in [JBB12]. This is done by representing the messaging middleware system as a finite state machine and verifying its constraints through model checking. The work checks that delivery guarantees are met as well as reconfiguration of the system after failures. Things like loops in routing tables are also checked for, to ensure that messages do not go through brokers through which they have already gone through. This works for cases where there are few brokers, or where the system can be broken down into smaller sub-systems, otherwise there are too many states to represent. The number of topics can lead to state explosion as well. As the current model does not consider time, the dynamic joining and leaving of publishers and subscribers cannot be modelled.

A publish-subscribe messaging middleware called Harmony is presented in [KKY$^+$10, YKK$^+$09]. It is QoS-aware and is targeted at wide-area networks. This messaging middleware supports performance requirements such as throughput and latency per message topic. This work evaluates the system using three different path selection techniques:

Proactive best-path Periodically search for the lowest latency path by probing paths.

Reactive QoS-aware Search for a better path when a latency constraint is violated.

Multipath Send topics across multiple paths.

Note that the multipath routing used in Harmony is used for reliability. Traffic is not distributed across multiple paths as in CAMPR, but is duplicated across them, sending the same data in parallel. Harmony aims to prevent QoS violations occuring due to delays caused by congestion through this duplication. The paths are ranked by latency and the two shortest paths are used for the multipath path selection technique.

Harmony uses a custom data transport layer called Tempore, which uses UDP multicast if available, otherwise unicast is used. The benefits with respect to end-to-end delay and message loss, and the costs with respect to traffic are discussed. The authors wanted to identify which path selection technique is suitable for which network conditions. Harmony was deployed in a wide-area air surveillance data distribution system and smart electric grids and was tested on a 5-node network testbed. Using multiple paths typically showed smaller end-to-end delay as compared to the other path selection techniques, but at the increased cost of more traffic, as the messages are duplicated. This technique also gives the least message loss. When the link delays in the network are quite variable, the reactive QoS-aware path selection technique is suitable, while the multipath technique is more suitable if packet loss is an issue.

2.3 Content-Based Networking

Content-based networks are another type of messaging middleware. In content-based networks, messages are routed according to their content rather than explicit addresses. As in traditional messaging middleware, the communication pattern in content-based networks is publish-subscribe. This sections discusses work done in this field.

Researchers from Palo Alto Research Center (PARC) argue that in the future an “Assurable Global Network” (AGN) will replace the Internet and this AGN will be “a secure content-centric network.”[JMSGLA07] They describe some properties of this type of network: [JMSGLA07]

Mappings This network will store mappings between names and data items and users will be able to retrieve the data item through a name query. Only the producer and those authorised by the producer will be able to change the mapping between names and data.

Attributes The data in the network will have attributes such as associated security keys or the type of data. Type information will allow the network to prioritise based on type.

Replication The data will be replicated across the network for resiliency and efficiency.

Integrity All data inside the network will be immutable, versioned and signed by its producer.

Since the data itself will be signed, it is not trusted based on the channel it arrived on. This means the data could be authenticated and delivered by anyone in the network.

Security Since the network will determine where information should flow on its own (administrators will not be able to input control information into the network) there will be no way to manipulate information flow accidentally or maliciously. There would be no need for so many layers of management infrastructure, for example DNS, which is a security risk as it is unauthenticated.

Feedback The network will collect feedback about the data in its nodes. For example if a malicious node sends "bad" information in response to a query and the receiver of the data detects this, this information will flow back along the path it came from, informing nodes along the way that the malicious node should be excluded from transmitting any further information.

This team proposes deploying this AGN as an overlay on existing IP networks, but wants to eventually replace them with a native routing layer. They have a basic 3 year plan of how to do this: [JMSGLA07]

1. Develop the routing and key distribution protocols for use in the content-centric network;
2. Deploy a testbed with applications running on top of the content-centric network and seek out users of the testbed;
3. Gather feedback from use of the testbed and iteratively improve its design and algorithms based on this feedback.

Lately there has been progress on this protocol described in [JST$^+$09]. A homepage has been created for Project CCNx[3] where the specification, documentation and implementation is available. There are two types of messages in the protocol: `Interest` and `Data` messages. They

[3] `http://www.ccnx.org`

are transmitted as efficient binary representations of XML. A comparison of the CCN stack versus the TCP stack is shown by downloading a file to multiple sinks. As the number of sinks increased TCP's download time increased linearly, but CCN's time remained constant.

To show that conversation-style communication (such as that done in IP networks) can still be done in this type of network, the team has implemented Voice over IP (VoIP) for this type of network: Voice over Content-Centric Networking (VoCCN). This is described in [JSB$^+$09]. Beyond emulating VoIP, this application can route calls to all destinations where they are likely to be answered.

Subbiah et al. propose a content-based network that works on top of multicast, however issues of scalability are mentioned and no concrete implementation or benchmarks are provided [SU01]. However they do mention the need for a "Content Broker."

In [CCK$^+$08] Cho et al. discuss applications of content-oriented networking. The paradigm shift towards content delivery is mentioned, citing peer-to-peer networks as an example where users are not interested in the source of the data but only the data itself. The inconsistency between the Internet architecture and its use is mentioned, and Cho et al. propose rebuilding the Internet to reflect its usage style more closely. Performance improvements and increased flexibility are mentioned as advantages of a content-oriented network. The closest source to a request that satisfies it can respond with the data, and also content-oriented networks allow for an easier implementation of multicast. The main advantage presented is that users of a content-oriented network have greater data accessibility and do not have to spend a lot of time searching for the information they need. They mention that they have started work on this network architecture.

In [BMVV05] Baldoni et al. propose an architecture that allows for a publish/subscribe system as an overlay with a novel multicast primitive which is more efficient than the standard unicast primitives used in such networks. They associate a *key* with each node in the overlay network and perform, routing based on this key. Multiple keys can be bound to the same node and the algorithm for the multicast primitive relies on this. The benchmarks provided do show an improvement with respect to the number of hops required to deliver messages over traditional unicast.

In [DGD05] Duan et al. discuss the design of network protocols in regards to traffic push or pull and how this relates to unwanted traffic. They compare the *sender-push* and *receiver-pull* traffic delivery models. They cite SMTP as an example of the sender-push model and HTTP and FTP as examples of the receiver-pull model. The general conclusion is that receiver-pull is preferred because it puts the receiver in control of the data they receive and when they receive it. Also the data will be exactly what the person wants and they are more likely to trust it if they themselves expressed interest in it. This prevents unwanted traffic such as spam.

In [SL04] Song et al. discuss a new system called “CBRBrain” to perform content-based routing over the Internet backbone. This is contrasted with peer-to-peer networks where each member of the network acts as a router. Several disadvantages of doing routing at the edges of a network (as in P2P networks) are presented:

Not all hosts are equal however all hosts have to perform the same functions despite widely differing computing power and available bandwidth;

Dynamic topology of end hosts frequently changes and this has a high maintenance overhead for routers;

Selfish hosts can deny relaying data to other hosts to save bandwidth and thus act as chokepoints.

The CBRBrain system basically relies on pure routers and separates hosts from routing, finding the desired content on behalf of users and then connecting them anonymously or giving them enough information so that they can connect to each other. As the network is made up of pure routers, the topology rarely changes and selfish hosts are not a problem since they are not doing any routing themselves.

In [CCP08] Castelli et al. propose a system called HyperCBR that would allow large scaled content-based routing. They argue that most content-based networks are based on a tree-shaped overlay network, with content subscriptions being broadcast; such an approach does not scale. Their method involves routing subscriptions and events on different partitions of the network space, which is divided into a multi-dimensional grid. For example, in a two-dimensional grid, subscriptions could be routed on the rows and events on the columns. They apply their implementation to content-based search in peer-to-peer networks and content-based publish/subscribe. Benchmarks were performed against Siena[4] and show that HyperCBR outperforms it, not just in very large scale networks, but also in smaller networks with less than 1000 nodes.

This same team of researchers have developed a model of the cost of content-based communication described in [CCP07]. They claim the model approximates simulations for networks with over $100,000$ nodes to within 3% and in most scenarios to below 1%. This allows researchers to not have to wait for results from simulations and can instead test their protocols based on the model.

In [RFH$^+$01] Ratnasamy et al. describe a scalable content-addressable network (CAN). They propose applying the idea of hash tables to all of the data on the Internet, citing peer-to-peer networks as an example of networks that could be improved by this technique. Each node of the CAN will store a *zone* of the entire hash table and requests to insert, delete and lookup data are routed towards the node whose zone contains the hash key of the data. The authors cite routing with over $260,000$ nodes with less than twice the IP path latency and are currently building a file sharing application that will use CAN.

In [JF08] Jerzak et al. propose the use of Bloom filters for routing. They implement their algorithm on top of the publish/subscribe platform Siena. Bloom filters are used to determine whether a message matches a predicate. The idea behind Bloom filters is that an array of m bits can be used to represent n elements. There are k hash functions which take an element

[4] `http://www.inf.unisi.ch/carzaniga/siena/index.html`

and produce a hash of index positions. Adding an element means setting the k bits of the array indexed by the hash of the element. Bloom filters can result in false positives (but not in false negatives), but the implementation presented produces an acceptable false positive rate. False positives would occur when elements have the same k hashes. To check whether a predicate matches a message, pass the message to the k hash functions and check that the bits of the array at all the indices of the hash have been set. If any of the positions in the array are 0 then the message does not match that predicate. Another optimisation proposed is to eliminate content-based addressing at all but the incoming router or directly at the publisher. For a different predicate matching algorithm see [Bit06]. The implementation has been made available as XSiena.[5]

In [EEM04] Eliassen et al. investigate how content-based networks could be used to deliver streaming video while providing receivers with fine-grained control over the videos they can subscribe to. Video publishers send data for every frame of the video thus allowing users to receive only the parts of the video in which they are interested.

A content-based routing (CBR) protocol made by modifying the Siena pub/sub system is described in [BBQV07]. The brokers are organised such that those with similar interests are close together. The overlay organises itself when subscriptions change taking network latency into account. So it induces topology changes rather than reacting to them. This is evaluated using J-Sim[6] and various distributions of events (publications and subscriptions). This protocol provides up to 33% reduced event transmission cost (in terms of hops to subscribers) compared to default Siena routing.

In [VBB03] Beraldi et al. describe a self-organising overlay for CBR which groups brokers with similar interests together. The CBR protocol described here uses Siena and has some interesting characteristics. A measure of similarity between subscriptions is defined, called associativity. The subscriptions hosted on brokers are called zones and on a new subscription the zone increases, and on an unsubscribe event it shrinks. Associativity between brokers is the

[5] `http://wwwse.inf.tu-dresden.de/xsiena`

[6] `http://sites.google.com/site/jsimofficial`

overlap between their zones. A broker's overall associativity is the sum of the associativities with each of its neighbours. When a broker receives a new subscription, this is an opportunity for it to increase its associativity by connecting to a new neighbour (and disconnecting from an old one if necessary to maintain an acyclic network). If a broker's zone decreases, this process is applied to one of its neighbours.

The protocols described in this section are slightly different from CAMPR, in the sense that subscriptions are not treated as queries for content sent over the network. An exception is PARC's content-centric networking protocol. The other protocols more closely resemble RSS usage patterns, whereas the protocol we are building is used in a way that is similar to a Google search query. Nevertheless, the fundamentals are still the same, messages are matched at each node to users' subscriptions expressed as predicates. That is the forwarding is unchanged, however the way forwarding tables are constructed, the *routing*, is different. Just like traditional networks, these types of networks also face congestion problems. Some schemes that attempt to alleviate congestion in networks are now disussed.

2.4 Congestion Control

Multicast congestion control in overlay networks is a simpler problem than congestion control in general. This is because in overlay networks multicast is performed by using the unicast services provided by the network. This amounts to congestion control between adjacent nodes in the overlay network. It has been shown in [UKB02] that overlay multicast congestion control schemes outperform end-to-end multicast congestion control schemes. The throughput does not decrease as the number of receivers increase. In this section, congestion control in overlay networks and a particular technique called backpressure are looked at.

The crudest way of determining congestion in a network is to measure packet loss. As an upgrade to this method, various tools have been developed for measuring the capacity and available bandwidth of links [PMDC03]. These tools usually rely on some form of probing packets, which can be expensive. Some routing protocols rely on this type of information to determine how congested a network is [LBS$^+$08, ZDA06].

One particular congestion control protocol presented in [PB03] also uses this type of information. This protocol was implemented within the Gryphon pub/sub system.[7] This protocol does not use packet loss to detect congestion. It adjusts the sending rate to that of the slowest link or broker in the network. Note that it does not adjust rates based on the slowest subscribing application, because this could potentially be slowing down the network maliciously. The protocol uses feedback from publishers to subscribers. The paper shows a scenario in which the lack of congestion control causes subscribers to stop receiving messages because of queue overflow. The broker which experienced the overflow asks the publishers for retransmission, but many of the retransmitted packets are also lost. To solve this, the protocol has two types of congestion control: limiting the publishing rate and limiting the rate at which subscribers request retransmission of lost packets. Both types were added as extensions to Gryphon's guaranteed delivery service. The protocol was evaluated by simulating network failures by disconnecting brokers and simulating congestion by limiting the available bandwidth on links. Results show that queues remain small while the protocol prevents congestion from building up.

Central to these types of schemes is having an accurate method of measuring the available bandwidth of links. The problem with using available bandwidth as a congestion metric is that it is expensive to calculate and worse, most methods are inaccurate. Another problem with using this metric is that less powerful nodes attached to fast links will get lots of traffic routed through them and as a result may become overwhelmed. The lack of processing power at nodes is something that bandwidth or capacity measurement techniques cannot take into account and so, if this is the only metric used, congestion can still occur at these nodes.

Backpressure is another type of congestion control used in overlay networks. It is a hop-by-hop mechanism which provides feedback when links become congested to notify senders to slow down or stop transmission. Some protocols have been proposed making use of this type of

[7] http://www.research.ibm.com/distributedmessaging/gryphon.html

congestion control [NT99]. Backpressure can also be combined with an end-to-end congestion control scheme [ZT10].

In [KGL00] Karol et al. describe a protocol that uses backpressure to prevent networks from dropping packets at congested times. These types of networks often have deadlocks and livelocks. The authors prove that their protocol is deadlock- and livelock-free. They define the following terms:

deadlock-free A network is deadlock-free when there are arbitrary packets in its buffers and these are guaranteed to be delivered in a finite time if no new packets arrive.

livelock-free A network is livelock-free if when arbitrary packets arrive into the network, and there are packets in its buffers, all packets are guaranteed to be delivered in a finite time.

The technique used in the protocol is called selective backpressure. When the network is not congested, all packets at a sender are eligible to be sent and can be selected by the scheduling algorithm. When the receiver's buffer fills up, this protocol restricts the set of packets that are eligible to be sent This is done by the receiver sending transmit feedback back to the sender. This allows the protocol to work with other protocols because the original packets do not have to be modified.

Basically there are classes of packets that may be sent out to a particular destination. Nodes maintain tables of the level of traffic they are allowed to sent to a particular node. A packet may be sent to a destination if the level of the packet is not less than the level stored for that destination. This level may change from hop to hop as the packet travels through the network because when a node receives a packet for a particular destination, the packet's level becomes that which the node has stored for that destination or one plus the most recent transmit feedback value received (whichever is greater). Thus a packet's class can be changed as it flows through the network.

The technique described here works on a hop-by-hop basis. It is based on buffer availability at the receiver, not on available link capacity. The buffer is divided into the classes corresponding to the levels. There are D levels where D is the maximum hops a packet may travel through the network. The idea is that the transmit feedback value sent by receivers is determined by buffer availability. The transmit feedback value is j where j is the highest buffer number (or level) that has enough room to store a packet with the maximum packet size used in the protocol. If such a buffer level doesn't exist a transmit feedback value of zero is sent.

This technique deals with short-term congestion overflows and deadlocks. The authors suggest combining their approach with a end-to-end congestion control scheme to solve this.

In [ZT10] Zolfaghari et al. describe a hybrid end-to-end and backpressure approach to congestion control. Congestion control algorithms are classified according to where the control decision is made. Backpressure is a hop-by-hop mechanism. It reacts to congestion on a short term time scale. In end-to-end congestion control algorithms, the source decides how to act. Congestion information is fed back through the network (this can be explicit or implicit).

[ZT10] lists some of the disadvantages of backpressure-based networks:

- lack of fairness
- unnecessary spread of backpressure feedback to more than one hop in the reverse direction
- possibility of deadlocks

Another problem mentioned was that when using backpressure networks, you need to maintain state at each intermediate node for each flow and this does not scale.

The protocol designed in [ZT10] was meant for connectionless networks. The backpressure in this protocol handles short-term congestion from traffic bursts while the end-to-end congestion control handles the longer term behaviour of the network. The algorithms work independently and the backpressure mechanism does not interfere with the end-to-end scheme.

[ZT10] states that to evaluate congestion two things are necessary:

- For each link, there must be a way to observe the congestion on an ongoing basis
- A way to transfer this information to sessions (series of packets) traversing the link

The algorithm does this in an interesting way, instead of using local information available to the routers such as queue length, sessions are designed such that they communicate their rate to the links in their paths. This information is used to estimate the flow at each link.

When arcs are added to the network, their capacity is calculated. This remains constant. When messages are generated, their size is known and this is the only thing affecting the residual capacities of the links. Therefore a source will know the amount of flow it has to send through the network. A source can maintain the amount of flow it has sent through a particular path (perhaps until acknowledgment is received from the destination of the message) and use this to figure out the path to send a message. The source assumes that it is the only user of the link and therefore no actual measuring of flow is needed as it's already known, it just needs to be transmitted through the network.

Noureddine et al. provide an analysis of backpressure schemes used in LANs in [NT99]. They split the schemes into three components: congestion detection, notification and control actions. Nodes must monitor the network and determine when congestion has occurred and also when it has ended. When a node is congested or is no longer congested it must notify other nodes. The paper mentions three types of notification schemes. In a simple scheme the flows involved in the congestion are not distinguished. In a class-of-service-based scheme nodes notify others about the class of the congested buffer, so that only this class is affected. In a destination address-based scheme the MAC address of the destination for the flow is communicated to other nodes. The control action considered here is the stopping and resuming of transmission until an explicit cancellation message is received.

[NT99] mentions some of the disadvantages of the flow-control performed by TCP such as unfair bias against bursty sources and expensive timer-based congestion detection. Backpressure can be used to avoid these, but also some scenarios are identified in which backpressure degrades the performance of the network. The authors argue that flows need to be distinguished and this calls for the MAC address backpressure technique described above. This also needs to be combined with the traffic class information so that different traffic classes do not slow each other down.

The problem with backpressure schemes are that they require significant overhead. Nodes need to maintain state for each receiver of each flow. Backpressure also adds unnecessary control traffic and sometimes does not improve network performance. Another factor is that backpressure is difficult to implement in such a way that it will never cause deadlocks or livelocks [KGL00]. Another problem with current congestion control schemes occurs when packets are dropped due to congestion and a node signals multiple senders for retransmission. This will usually result in dropped packets again as they will be retransmitted simultaneously.

For these reasons, other ways of attacking the congestion problem were looked at. One technique used to increase throughput is sending messages along multiple paths between a source and a destination. Some work done in this field is now looked at. In Table 2.1 a summary of the features discussed in the following sections is presented. This compares protocols for multipath routing, traffic engineering and bandwidth reservation with CAMPR.

2.5 Multipath Routing

Current overlay routing protocols calculate a single shortest path between source and destination pairs. Paths with fast links will be used disproportionately more than the buffer space along those paths allows. Available network resources for a message flow are constrained only to those available along the shortest path. This leads to congestion and a decrease in throughput. Note that much of the early work done in the field of multipath routing tends to focus on reliability and error recovery, thus the term multipath routing in that context tends to mean using multiple paths to send the same data for increased redundancy, rather than splitting the data across multiple paths to increase throughput or avoid congestion. In this section, multi-

	Distributed	Load-balance	Min. Usage	Disjoint Paths	Msg. Splitting
[WWK$^+$07]	✓				
[MFC08]				✓	
[IUY06]			✓		
[IN02]		✓		✓	
[BO07]		✓		✓	
[XJT09]	✓				
[MP09]	✓				
[NZ01]			✓	✓	
[AMG05]	✓			✓	
[EJLW01]	✓		✓		
[HBCR07]	✓				
[KKDC05]	✓		✓		
[ER04]			✓		
[SWW$^+$04]	✓			✓	
[ABIS06]	✓				
[SKY11]					
[ABKM01]	✓				
[HWJ05]				✓	
[LG01]	✓			✓	
CAMPR		✓	✓	✓	✓

Table 2.1: Feature comparison of the protocols discussed in Sections 2.5, 2.6 and 2.7

path routing as used in overlay networks, to divide rather than replicate traffic across multiple paths is focused on.

A way to overcome this is to allow for messages to be split across multiple paths from a sender to a receiver – multipath routing. This technique can alleviate congestion by providing more resources for a message flow, thus increasing throughput. It has been shown in [HSH$^+$06] that a multipath approach can be combined with a congestion control scheme to achieve greater throughput than that possible through a single-path approach. Much work has been done on multipath routing and various protocols have been proposed. In [OIM09] it has been shown that generating many alternative paths can be efficiently done. An improved protocol is presented in [CER12] which also allows for link failures.

This type of routing in the context of the application layer is talked about. There is also work on multipath TCP, but this field deals with other problems which are not relevant to this work. For example, in order delivery and deployment issues. Splitting traffic across multiple overlay paths, with standard TCP running on each overlay path is focused on. To the best of my knowledge there are no multipath routing protocols that deal with message splitting (see definition 3.3).

A distributed approach to multipath routing is presented in [WWK$^+$07]. This uses a heuristic to iteratively select paths and control the sending rates along the selected paths. This approach has a different goal from most approaches: it aims to maximise the sending rate of sources rather than minimum network usage or load-balancing. Sources probe paths for available bandwidth and accordingly decrease or increase their sending rates. Source nodes use local information to do this, so the algorithm is distributed. Due to complexity, two randomised methods for selecting paths are used. In the first method, sources randomly choose k paths. No knowledge of the network is required for this method. In the second method, paths are chosen with a probability proportional to the available bandwidth in the path. If demand requirements are not met, paths are iteratively reselected until the requirements are met.

In [MFC08] a routing protocol which uses the Ford-Fulkerson algorithm is proposed. This protocol calculates disjoint paths between source and destination pairs, ranks them according to length and then distributes traffic among them in a round-robin fashion. As network load increases, less end-to-end delay is observed using multiple paths versus using single paths. This approach is then combined with a congestion detection mechanism in [MCM09].

An interesting approach to multipath routing is presented in [LBHO05]. The algorithm presented has similar computational complexity to Dijkstra's algorithm. All paths between source-destination pairs are computed and then packets are routed with a certain probability across these paths. The algorithm computes the next-hop probabilities. This approach partitions the network to reduce complexity. Nodes work with incomplete topology information to make routing decisions. The authors argued that by partitioning the network into components which are well connected, this does not reduce the quality of the routing as compared to protocols which have full knowledge of the topology.

In [IUY06] a multipath routing idea is presented which allocates paths in a similar way to this work. The path distribution problem is treated as an optimisation problem. A model is generated and a solver is used to allocate bandwidth according to two goals: link usage and allocating bandwidth fairly. The work is evaluated on small topologies, the largest one has only 24 nodes. The model used with CAMPR scales to larger networks. This work also uses link bandwidth to constrain the model variables.

A model similar to one of the original optimal models is presented in [IN02]. This paper mentions the fact that these models are very slow to solve, even for tiny networks. Also the models do not accurately portray the network state, as rounding and truncating has to be performed, leading to errors in flow allocation. The authors propose a heuristic approach which approximates the solution of the models. This was one of the problems faced with initial models. See Section 3.6.1 for more details.

In [BO07] it has been shown that treating the multipath routing problem as an optimisation problem can give more efficient results than a heuristic approach. A distributed approach to multipath routing is presented in [XJT09] which also uses an optimisation technique. In this approach nodes can make routing decisions locally without knowledge of the entire network topology. This approach is evaluated on topologies up to 200 nodes. Another distributed approach is shown in [MP09]. In this work shortest paths are ranked according to available capacity in the links. Using this approach in CAMPR, the results were not very good. See Section 3.5 for more details on path selection.

The multipath routing scheme presented in [NZ01] uses a hybrid approach for traffic allocation. Paths are selected based on global information, but traffic is divided among them using only local information about available bandwidths in the paths. Although traffic is split among paths differently, the idea to use bottleneck-disjoint paths in CAMPR comes from this paper. For more about the path selection technique see Section 3.5.

In [ABKM01] a resilient overlay network (RON) is described. The focus of this work is on reliability. RON probes and monitors paths, checking their quality, considering latency, packet loss rate, and available throughput. A link-state routing protocol is used to disseminate this information. The authors claim that RON can usually route around failures using just one intermediate hop. The secondary goal of RON is to integrate path selection closely with applications. This allows RON to use application-observed throughput to select paths. Applications can choose a single metric to influence path selection: latency, packet loss or throughput. RON does not try to select optimal paths with respect to throughput, but instead tries to avoid paths of low throughput. The authors claim that is because available bandwidth is unpredictable and changes too quickly. However, the main focus of RON is on routing around outages and recovering from failures. Most of the experiments evaluate how quickly RON can recover and route around failures.

Some of the authors of [ABKM01] further evaluate multipath routing in [ASB03] using the RON testbed, however the focus of this work is also on reliability when network failures occur. The multipath routing described here is simply for increased redundancy, routing the same content across multiple paths. This is different to the approach used in CAMPR, which focuses on avoiding congestion by splitting content across several paths.

A topology-aware overlay network is described in [HWJ05]. The focus of this work is overlay node placement such that overlay paths can provide the best performance through paths which are disjoint with respect to the underlying network. This overlay node placement is an offline process, using path diversity and latency as metrics for node selection. The authors focus on recovering from path outages and routing around network failures. Although the focus of this work is different from CAMPR, the authors identify an important concept which is applicable to CAMPR as well: the effectiveness of the overlay paths depends on how disjoint they are. This was felt to be a critical issue in CAMPR and bottleneck-disjoint paths were chosen to provide more bandwidth for routing.

In [LG01] a multipath routing protocol for ad hoc networks is presented. This protocol is called Split Multipath Routing (SMR) and it uses maximally disjoint paths to route traffic. Similar to the path selection technique of CAMPR, one of these path is the latency-shortest path. SMR limits the number of selected paths to two. The authors mention that due to packet reordering issues SMR does not work with TCP. This is used in ad-hoc networks where connectivity changes frequently and is an on-demand process. The focus is again on recovering from path failure rather than congestion avoidance and the two types of path selection techniques evaluated reflect this. The techniques are generating a new pair of paths when either of the existing paths disconnects and generating a new pair of paths only when both of the existing paths disconnect. According to the results shown, the latter scheme is more effective.

Now, another field which uses multipath routing, traffic engineering is looked at.

2.6 Traffic Engineering

Another field in which multipath routing is frequently used is traffic engineering. Now, some of the work done on multipath routing in this context is looked at. Usually the goal of these schemes is similar to the goal of this work, reducing or avoiding congestion and load-balancing.

In [AMG05] a routing algorithm similar to ours is presented. This algorithm calculates all possible delivery graphs between a sender and its set of receivers, ranking them according to their bottleneck residual bandwidth. The algorithm tries to avoid bottleneck links, although in a different way from CAMPR. Once all delivery graphs are calculated, the number of times a link appears in the delivery graphs is counted. Past a certain threshold, a link is considered to be a probable bottleneck. The links' residual bandwidths are monitored and past a certain threshold they become bottlenecks which are avoided when splitting traffic across delivery graphs. This algorithm also uses a modelling approach to routing. The authors claim that the algorithm finds the minimum number of paths required to satisfy the bandwidth requirements. Finding the minimum number of paths is the objective function of the model. The algorithm is evaluated by measuring the number of admitted requests. This algorithm relies on measuring available bandwidth, which cannot be done cheaply and accurately. This approach also leads to oscillations in path selection with the changing residual bandwidth.

The traffic engineering approach presented in [EJLW01] also performs multipath routing using optimisation to balance load across the paths. This technique uses probe packets to monitor congestion in the network. The measure used for congestion is packet delay and packet loss. The paper states that available bandwidth would be a better metric, but it is too difficult to measure accurately. Probe packets are sent from senders to receivers and then from receivers back to senders. Intermediate nodes do not participate in the measurement. Statistics are collected and the load is shifted accordingly. The authors claim that according to Internet measures, aggregate traffic on links is relatively stable in five minute intervals. This is an online process, however it is reactive in the sense that it waits for packets to be dropped before reallocating traffic. The authors assume that paths between senders are receivers have already been calculated. This approach was simulated using the model on very small topologies and there is no implementation.

A similar approach is presented in [HBCR07]. This approach focuses more on the throughput of users, but it does try to achieve both goals, low congestion and high aggregate utility. The paper claims that just maximising user utility is not a good goal in the long term, and that the joint goal makes the network more stable. The model presented is constrained by the link capacities, and the approach is to limit sending rate once this is reached. The authors point out that bottlenecks in the paths are what matters. This model is also tested on very small topologies. There is no implementation, but one of the design goals of the authors is to make deployment feasible by requiring changes to as few routers as possible. This dual approach to maximise throughput and minimise congestion is rare in the traffic engineering world.

A distributed online traffic engineering approach is presented in [KKDC05]. This approach also performs multipath routing. Its path splitting approach was borrowed from the work in [EJLW01]. Like most traffic engineering schemes, this one also aims to minimise the maximum usage of links. The k-shortest paths are calculated offline between every sender-receiver pair. These are ranked according to propagation delay. The paths selected are not link disjoint and are rarely recomputed. Each sender probes the paths available to it to measure their utilisation. If probe packets are dropped, the sender increases its estimation of the path utilisation and the load is balanced accordingly. This protocol was compared against the online protocol presented in [EJLW01] and the authors claim that it is more effective at load-balancing.

The common approach of most of the work described in this and the previous section is to model the network and the demands placed on it by the users as an optimisation model which can be solved by a linear programming solver. One approach is to describe the optimal model and then create heuristics that aim to estimate the solution to this model because solving the optimal model is intractable. Another approach is to find ways to simplify the optimal model making it tractable. The latter approach was taken in CAMPR, discussed in detail in the next chapter.

2.7 Bandwidth Reservation

In this section, bandwidth reservation techniques are discussed. The field of bandwidth reservation also aims to make the most of network resources in order to meet user demands. This QoS technique can be used in conjunction with traffic engineering and multipath routing.

A bandwidth reservation protocol which uses multipath routing is presented in [ER04]. The work is presented in the context of providing traffic guarantees to users of a VPN. This paper takes a modelling approach to determine bandwidth allocation. The model is different from most in that both outgoing and incoming traffic are represented in the model. This model assigns a value between 0 and 1 to each edge, representing the fraction of traffic to be routed through the edge. Each edge is represented in the model and therefore it can only be solved for very small topologies. The paper shows experiments with topologies of up to 100 nodes. The authors argue that since VPN connections are established for long periods of time, it is worth spending time to calculate the bandwidth allocation. This paper shows that it is advantageous to use multipath routing over single path routing.

Another multipath bandwidth reservation scheme is presented in [SWW$^+$04]. This work is in the context of ad hoc networks where bandwidth is constrained. As there are no fixed link connections, the bandwidths considered are related to the nodes. This bandwidth reservation approach splits bandwidth requests into smaller requests among multiple paths. As in CAMPR, this approach prefers selecting disjoint paths. Traffic is distributed among the selected paths in proportion to the available bandwidth. The problem is not treated as an optimisation problem. The approach is simulated with up to 50 mobile nodes and is compared to a single path approach. By splitting the bandwidth requests into smaller requests, the probability of successfully being able to reserve the requested bandwidth increases. The authors mention that with more than 30 nodes, there are not enough resources for multipath routing.

A dynamic bandwidth reservation technique is discussed in [ABIS06]. The approach used is to estimate future traffic demands based on previous traffic measurements. The scheme attempts to minimise the amount of excess bandwidth reserved while at the same time minimising the risk of congestion occurring by reserving just the right amount of bandwidth required by users. The authors mention that in some cases it is possible to know in advance the required bandwidth and cite on-demand video as an example. Experiments are performed on a topology of less than 20 nodes and results show how close the bandwidth reservations based on the predicted traffic demands are to actual traffic demands.

A heuristic approach to bandwidth reservation is discussed in [SKY11]. The algorithm tries to iteratively reserve network resources for all the requests by attempting to reserve bandwidth for some of the requests, updating the available bandwidth and repeating with the remaining requests. If an iteration does not allow for any new reservations, then the algorithm stops and any remaining requests cannot be satisfied. This algorithm takes into account a time limit for each request and tries to minimise the total time needed to complete transfers. The algorithm runs in polynomial time. It is compared against a first come first serve approach in which reservations are attempted in the order in which they are received, and a largest bandwidth first approach in which reservations are attempted in decreasing order of required bandwidth.

2.8 Summary

In this chapter, the foundations of this work were described. Some fields related to this work were looked at: messaging middleware, content-based networking, congestion control schemes, multipath routing, traffic engineering and bandwidth reservation. Similar problems are faced in these fields, but these problems were brought together into a multicast multipath congestion-avoidance routing scheme with message splitting. This context creates new challenges which were attempted to be dealt with by modelling the network and looking at them as an optimisation problem. In the next chapter, the modelling approach is described in detail.

Chapter 3

Multipath Message Splitting Models

3.1 Introduction

Congestion-avoidance routing was approached as an optimisation problem. The state of the network, publishers, subscribers and the messages were represented as a mathematical model which can be solved using optimisation software. The solution to this model describes how to construct forwarding tables. In this chapter, the process of generating a model from these inputs is described. Doing this involved finding shortest paths from senders to receivers. Different ways of doing this was experimented with. Finally, the various models used to attempt to accurately model the routing protocol are formally described.

The context of interest is a store-and-forward network in which multiple senders (publishers) send messages to multiple receivers (subscribers). Each message may have many or no subscribers. The goal is to optimised the routing of the messages through the network. This means that receivers are to receive the messages to which they are subscribed as quickly as possible. This implies no dropped packets causing retransmissions, no rate-limiting and sending messages along the shortest paths, for various definitions of shortest. These things are done by efficiently using network resources in order to avoid overloading the network. If network resources are greedily used in a local manner, this leads to global inefficiency causing dropped packets and an overall decrease in throughput.

To avoid this messages must be distributed along paths which have the resources to handle them; to optimally distribute network resources to serve the demands imposed by the users. Doing this optimally on a network-wide scale is intractable, and some assumptions must be made, even to approximate an optimal solution.

Even a suboptimal solution to this problem takes time to calculate. The calculation time is assumed to be slower than the rate of publications, therefore the solution presented is an offline solution. However, this solution works well for long-running flows (subscriptions), which are not uncommon in publish-subscribe systems. Every message in the network is not reacted to. This means that capacity is taken into account, but residual capacity is not. The offline solution still reacts faster than current Internet routing which can take several hours, and for small networks or message loads may even be used as a real time solution.

Standard traffic engineering approaches to balancing load across the network are also slow to react to changes. As discussed in the previous chapter, there are indeed online solutions to this type of problem, however, they usually have other assumptions, such as relatively stable traffic volume. They sometimes assume that paths between senders and recievers have already been calculated and rarely change. These approaches also sometimes assume that available bandwidth can be accurately measured through probing or simply by measuring dropped packets, and rely on this as a metric to determine congestion. They do not lead to globally optimal solutions, but instead rely on using heuristics to give a local solution. They also usually have much simpler routing assumptions, such as no message splitting, no path splitting and no multicast.

Complete knowledge of the topology of the network is assumed. This is not a distributed scheme and it aims to optimise traffic globally. This is a common assumption in non-distributed schemes. Global knowledge of all messages in the network is also assumed. This means that it is known who is publishing a message and who its subscribers are. This type of assumption is also common in traffic engineering. In publish-subscribe systems, all nodes are aware of the

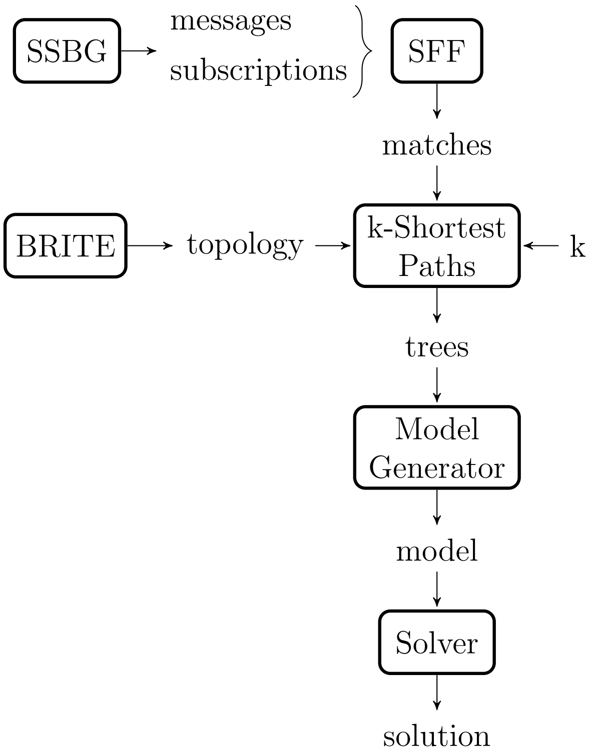

Figure 3.1: Model generation workflow

subscriptions in the network anyway. The only difference is knowledge of the publishers, so this assumption is not unreasonable. The approach to achieving these goals is now described.

3.2 Approach

The approach taken to achieve the goals discussed in the previous section is not described. As mentioned previously, this was cast as an optimisation problem. The state of the network, senders, receivers and the messages are represented as a mathematical model which can be solved using optimisation software. The state of the network includes things such as the topology and the latency and capacity of its links. The solution to this model describes how to construct forwarding tables.

In [BO07] it is shown that treating the multipath routing problem as an optimisation problem can give results which are closer to the optimal solution than those given by a heuristic approach.

Before treating this as an optimisation problem, MATLAB[1] was used to attempt to calculate solutions to these problems, but this proved to be too slow to be of any practical use. For these reasons a linear programming model and a linear programming solver were chosen over a heuristic approach.

Relying on a model also has disadvantages when compared to a heuristic approach. The obvious overhead is that the network state needs to be modelled and also a model solver needs to solve the resulting model. This overhead does not exist in a heuristic approach, which is more suitable to an on-demand routing solution, especially in networks with frequently changing traffic patterns. There may also be no solution to the model if there is insufficient bandwidth in the network. The model solver does not give a next best or almost feasible solution, but using a heuristic approach, perhaps a more elastic solution can be obtained, even in the case of insufficient resources. A heuristic approach can be more flexible, albeit further from the optimal solution.

An overview of the steps involved in generating a model to represent a given network state and workload are now given. For a start, assume there is a topology, message workload and senders and receivers for these messages. As it is not essential to know the details of these parts of the process when discussing the model, the generation of these things is discussed in more detail in the next chapter. They are included in the workflow diagram shown Figure 3.1 for completeness. The workflow used in this approach is now described.

A technique that can be used to increase throughput is to split traffic between a sender and a receiver along multiple paths [BO07, WWK$^+$07]. This technique, discussed in the last chapter, is called multipath routing. Splitting traffic in this way can also be used to reduce congestion, as more resources become available for a given flow of messages. Less traffic has to be sent along a given path as all paths share the burden of the flow.

For this type of routing to be effective, flows have to be split carefully so as not to overload the network. Depending on the goal, different splitting schemes have to be used. Different ways to

[1] http://www.mathworks.com/products/matlab/

split flows across paths were experimented with. For a sender sending a message to multiple receivers, there are multiple paths going to each receiver. The paths from a sender to a receiver are the k-shortest paths. The shortest-path algorithm used and the various ways paths were ranked are discussed in Section 3.5. One set of paths from sender to receivers is the delivery graph. There are multiple delivery graphs across which the message flows can be split.

Once the delivery graphs have been constructed for all the flows in the network this information is given to the model generator. This outputs a model which can be solved by optimisation software. In this case the Gurobi[2] solver was used as it is free for academic use. However the generated model can also be solved by CPLEX[3]. Originally, CPLEX was used for an earlier version of the model which took a long time to solve. Gurobi was preferred because it could solve this model faster. However, the present model is solved very quickly, so either solver could be used with negligible difference.

The pipeline of generating delivery graphs from a workload and then generating a model and solving it works iteratively. At first, $k = 1$, which means one delivery graph per flow. If no solution is found to the model generated from this, k is incremented, generating two delivery graphs per flow, and generate a new model. This process continues until a solution is found or until k reaches one hundred. At this point, it is considered that there is no solution. If a solution exists, it describes for each flow, how many messages of it to send down which of its delivery graphs. There are different models with different goals for distributing flows across delivery graphs. The models are described in detail in Section 3.6.

Something that is unique to CAMPR is message splitting. This allows a node which has received a single copy of a message to copy this message to outgoing buffers and send it to multiple neighbouring nodes. This reduces traffic in the network, leading to less congestion and greater throughput. Message splitting complicates routing decisions as splitting has to be done at the right time in order to reap the benefits of this feature. This is something else that is taken into account in the model of the network and to the best of my knowledge this is the

[2] `http://www.gurobi.com`

[3] `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`

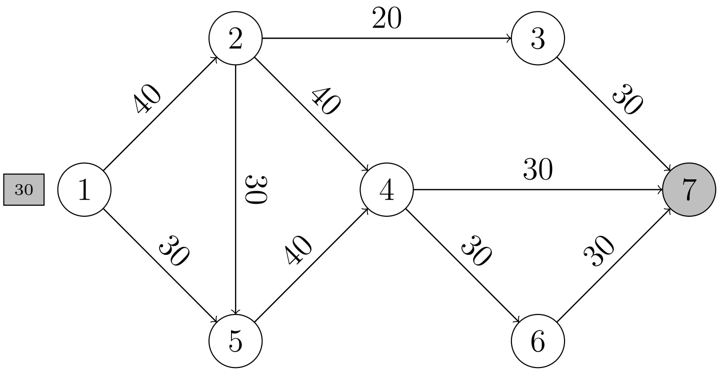

Figure 3.2: Example topology with capacities of links shown. Node 1 sends a flow of 30 messages to node 7.

first model to do so.

An example is now used to illustrate CAMPR. In Figure 3.2 a small topology consisting of seven nodes is shown. The labels on the edges are the capacities of the links. Node one wants to send a *flow* consisting of thirty *messages* to node seven. The latencies are not shown, but it is assumed that the latency-shortest path ($k = 1$) between the nodes one and seven is $1 \rightarrow 2 \rightarrow 3 \rightarrow 7$. In this case there is only one sender and one receiver, so the entire delivery graph of the flow of messages from node one consists of just one path. The flow of thirty messages cannot fit in the link $2 \rightarrow 3$.

In traditional overlay routing, the flow would be sent along this path despite there not being enough capacity. Packets would be dropped and then retransmitted. Despite this being the latency-fastest path, the retransmission of packets will delay the delivery to the destination. In CAMPR multiple paths between senders and receivers are used, giving more capacity for flows. As the flow cannot fit along the latency-fastest path, another path ($k = 2$) is requested. The bottleneck link in the first path is $2 \rightarrow 3$ and will therefore be avoided in subsequent paths.

Assume the second latency-shortest path is $1 \rightarrow 2 \rightarrow 4 \rightarrow 7$. The flow fits entirely along this path and therefore no further paths will be requested. The two delivery graphs are shown in Figure 3.3. The model generated using these delivery graphs has a solution. In this case the first path found will remain unused. There will be no path splitting or message splitting. This

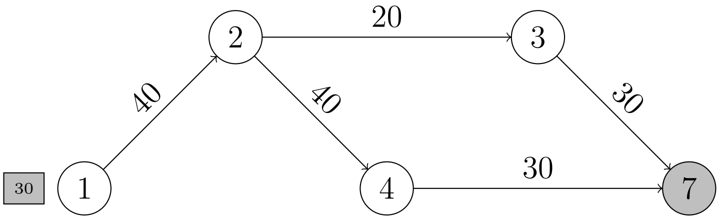

Figure 3.3: Two delivery graphs ($k = 2$) from node 1 to node 7.

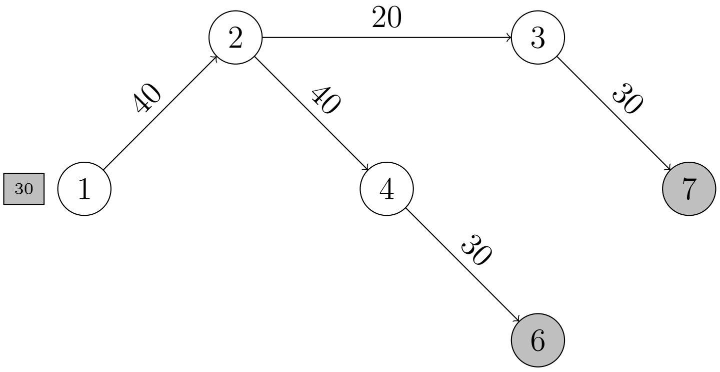

Figure 3.4: Delivery graph ($k = 1$) from node 1 to nodes 6 and 7.

is a minimum network usage solution. As few nodes as possible are used for routing the flow.

An alternative to this is to use *path splitting*. Suppose the flow is divided across the two delivery graphs such that fifteen messages go along the first path and fifteen messages go along the second path. In this case the two delivery graphs overlap at link 1 → 2 so all of the flow will be sent to node two. Once node two is reached, the path splitting begins. At this point fifteen messages are sent via node three and the remaining fifteen messages of the flow are send via node four. This is a load-balancing solution. Load is evenly distributed across the delivery graphs such that the maximum amount of flow sent along any link is minimised.

A more complicated example is now discussed. As before, there is one publisher, node one. However this time there are two receivers, nodes six and seven. Both subscribers want to receive the thirty messages published by node one. This time two paths are generated per k. The first path going to node seven will be as before, $1 \rightarrow 2 \rightarrow 3 \rightarrow 7$. The first path going to node six will be $1 \rightarrow 2 \rightarrow 4 \rightarrow 6$. This delivery graph is shown in Figure 3.4.

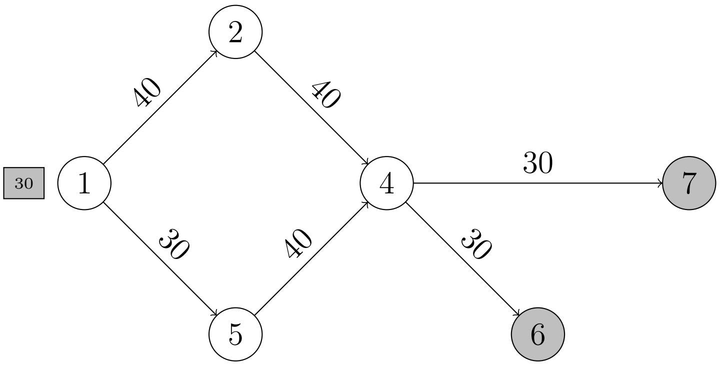

Figure 3.5: Delivery graph ($k = 2$) from node 1 to nodes 6 and 7.

As before, another delivery graph needs to be requested as the flow does not fit through link 2 → 3. Assume that for the second delivery graph the same path is calculated from node one to node seven as before, 1 → 2 → 4 → 7. The bottleneck link in the path between node one and node six is 4 → 6; however in CAMPR, if a bottleneck link is attached to a sender or a receiver, it is not ignored it in subsequent paths. This is to prevent the possibility of running out of paths and making the destination unreachable. Indeed, in this topology, if link 4 → 6 was avoided in subsequent paths, there would be no paths left going to node six. Assume the second path going to node six is 1 → 5 → 4 → 6. The second delivery graph is shown in Figure 3.5.

As before, one alternative is to use the minimum amount of network resources by using only the second delivery graph. This alternative has no path splitting. One way to route in this case would be to send all the messages to node four via node two. Once node four is reached, *message splitting* is used. Node four duplicates the thirty messages and sends them to nodes six and seven. This saves bandwidth as node one only has to send thirty messages instead of sixty to satisfy both receivers.

Another alternative is the load-balancing solution. Assume the flow is split in half as before across the two delivery graphs. In the first delivery graph fifteen messages would be sent to node two and then they would be split going via nodes three and four to the receivers. In the second delivery graph, in order to load-balance, the fifteen messages will be sent to node four

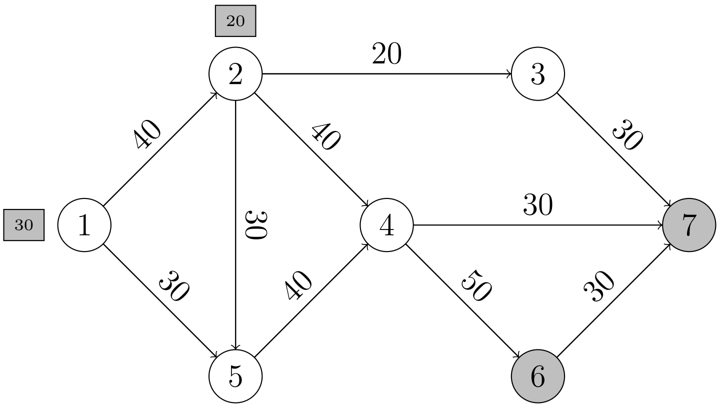

Figure 3.6: Example topology with capacities of links shown. Node 1 sends a flow of 30 messages to node 7.

via node five because links 1 → 2 and 2 → 4 are already being used in the first delivery graph. The messages would be split at node four.

An example with two senders shown in Figure 3.6 is now considered. The topology is the same as before, with node one sending thirty messages to nodes six and seven, but now node two also sends twenty messages to node six. Note that the capacity of link 4 → 6 has been changed to fifty to make this a feasible example.

Node one and its receivers have the same delivery graph as in the previous example. For node two, assume the latency-fastest path is 2 → 4 → 6. These delivery graphs are shown in Figure 3.7. In this case the flow from node two fits along the delivery graph. However, the delivery graph of node one must also be taken into account, which overlaps at link 2 → 4. Given the current workload fifty messages would need to be sent through a link with only forty capacity. Therefore using these delivery graphs is not feasible in this case and a second set of delivery graphs must be generated.

The second delivery graph for node one is the same as before. The delivery graph for node two is 2 → 5 → 4 → 6. These delivery graphs are shown in Figure 3.8. Given these two delivery graphs the flows can be routed. The minimum usage solution again would be for both senders to send all of their messages along the second delivery graphs. In this case node one would send all of its messages to node four via node two and the messages would be split at node four.

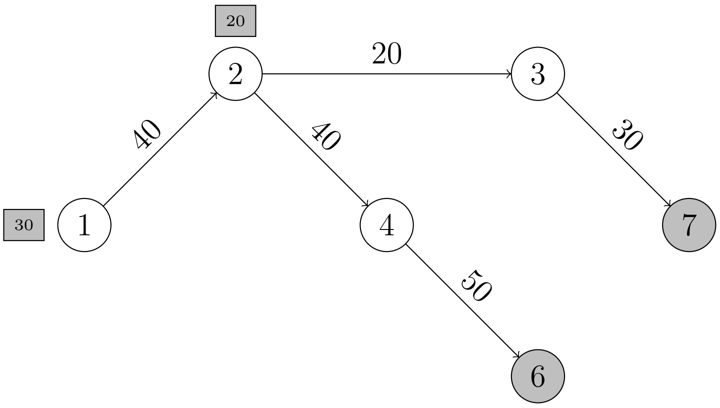

Figure 3.7: Delivery graphs ($k = 1$) from node 1 to nodes 6 and 7 and from node 2 to node 6.

Note that node one cannot send the messages via node five because the link 5 → 4 is shared with node two's delivery graph and both flows cannot fit along this link.

In the load-balancing case, node one would split its flow in half as before. In the first delivery graph, the fifteen messages would be split at node two. In the second delivery graph, the fifteen messages would be sent via node five and split at node four. Node two would also split its flow in half, sending ten messages down each delivery graph.

Hopefully the examples presented in this section have made the concepts clearer. In practice the topologies are much larger, there are more senders and receivers and more flows. However, the approach presented through the small examples can be generalised to the larger topologies and workloads. In the next section, some terms used informally to explain CAMPR are formally defined.

3.3 Definitions

Some terms used in the previous section and in the rest of this thesis are now formally defined. These definitions assume the context of a graph $G = (N, E)$ with nodes N and edges E of the form $l = (n_1, n_2)$ where $n_1, n_2 \in N$. A path is a sequence of nodes $p = (n_1, ..., n_m)$ where $n_i \in N, \forall i : 1, .., m$ such that $(n_i, n_{i+1}) \in E, \forall i : 1, ..., m - 1$. The capacity of an edge $e \in E$ is

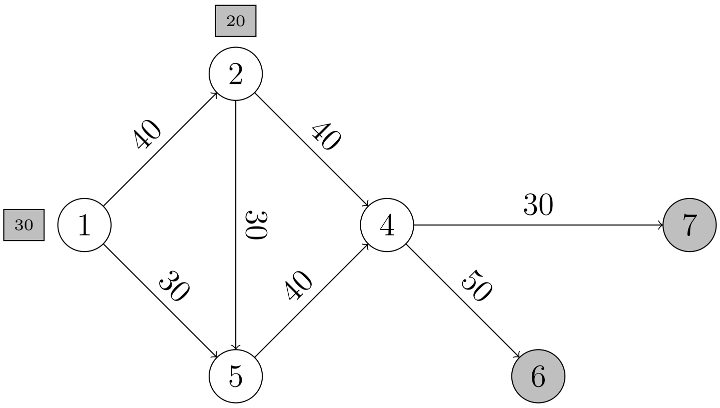

Figure 3.8: Delivery graph ($k = 2$) from node 1 to nodes 6 and 7.

c_e. There is a set of messages M.

Definition 3.1 (Flow) *A flow f is the set of messages $m_1, ..., m_n \in M$ which are sent from the same sender $s \in N$ to the same set of receivers $R_f \in N$. Flow f has size n. Each message has the same size. In sending a flow to its receivers, it may be split (at the message level) across multiple delivery graphs.*

Definition 3.2 (Delivery Graph) *The set of paths from a sender to the receiver(s) of a flow is a possible delivery graph of that flow. This is generated by finding the shortest path from the sender to each of the receivers and then combining these paths. For a set of paths $P = p_1, ..., p_n$ made up of the sets of nodes $N_{p_1}, ..., N_{p_n}$, a delivery graph D_f of a flow f consists of all the unique nodes in $N_{p_i}, \forall i1, ..., n$. A single node may be part of multiple paths in the delivery graph. The delivery graph is rooted at the sender and the leaves are the receivers. There are no loops in the delivery graph, but as paths contained in it may overlap, it is a directed acyclic graph.*

There are k such delivery graphs for each flow. These delivery graphs are then used to generate a model representing the network state.

Definition 3.3 (Message splitting) *Message splitting refers to the ability of a node $n \in N$ receiving or publishing a message $m \in M$ to duplicate m and send it to any amount of its* *neighbours, even though it has received or published only a single copy of* m*. When the message is sent out to more that one node it is considered split. This saves network resources by requiring less capacity. Splitting as close as possible to the receivers may require the least capacity, but when considering all the messages in the network, this may not be optimal or even feasible.*

Definition 3.4 (Path splitting) *A publisher publishing flow* f *of size* n *may choose to divide this flow into at most* n *parts, as this is how many messages are in* f*, and send each part along a different path to its receivers.*

3.4 Goals

The different goals to be achieved with CAMPR are now discussed. To give a more intuitive explanation, the goals can be described from the highest level as being maximum flow and minimum latency. The first goal can be roughly defined as routing as much as possible from one node to another node. However, in this case it is actually a multicast maximum flow; there are multiple receivers per sender.

The second goal is to route messages along the fastest links from senders to receivers. The other thing taken into account is achieving these goals not just for one sender without considering how this affects the other senders, but finding a *global* solution, taking into account all of the senders and receivers in the network. These goals are at odds with each other and often the best solution for each goal is different. Achieving a good balance of throughput and latency without sacrificing too much of either is a compromise. The meaning of these goals is now refined.

The goal of maximum flow can seem to be maximising the senders' sending rates however this greedy approach actually does not lead to good solutions globally. The way this goal is approached is with the sub-goals of load-balancing load across the network and minimising network usage. These goals are specialisations of the goal of just finding a feasible routing solution given a topology, senders and receivers and a message workload. A feasible solution might be neither load-balancing nor minimum usage.

The load-balancing goal means that for a given sender with a set of receivers and a set of k delivery graphs, the maximum amount of flow sent along each delivery graph is minimised. This implies spreading the flow as much as possible across the k delivery graphs. This is calculated globally, taking into account every sender and its receivers. The point of this goal is to avoid overloading any routers or links in the network by spreading load as evenly as possible. This is the most common goal used in traffic engineering schemes as it is important for ISPs to maximise throughput while requiring the least capacity possible in their networks.

The minimum usage goal means minimising overall network usage. This goal implies the opposite of the load-balancing goal. Load is distributed across as few delivery graphs as possible, while still fitting in the capacity of those delivery graphs, i.e., being a feasible solution. This goal is not common in traffic engineering as it is more user-oriented rather than ISP oriented. This implies that more load may be placed on individual links or routers using this goal.

The other overall goal is minimum latency. This is treated this as a secondary goal, with maximum flow being more important. If a latency-fast path does not have enough capacity to satisfy a demand, it would be rejected in favor of a latency-slower path with enough capacity. Nevertheless, this is still consider as the speed of delivering messages is also important. Indeed, sometimes receiving messages too late may be as bad as not receiving them at all. Therefore a balance must be reached between meeting capacity requirements and the speed of the paths. This goal can be further refined into objectives such as minimum hop and minimum cost.

These goals cover the most important objectives in delivering messages within a network. There might be other possible goals, but maximum flow and minimum latency are considered in this work. Meeting these goals comes down to selecting appropriate paths for message delivery. The other aspect is how to distribute messages across the selected paths for delivery. Both of these aspects must consider all senders and receivers and all messages in the network. Different attempts for selecting shortest paths for generating the delivery graphs are now discussed.

3.5 Path Ranking Techniques

The algorithm described in [dQVMdS00] was used to determine the shortest paths between senders and receivers. This algorithm has good time and space complexity, but the shortest-paths algorithm is independent of the model used and as better algorithms become available they can easily be swapped in. The algorithm runs in $O(km)$ time after the shortest paths tree from the source node has been calculated and $O(kn+m)$ space, where k is the number of paths, m is the number of arcs in the graph and n is the number of nodes in the graph. Experiments were carried out with different ways to rank paths. There are now described.

3.5.1 Capacity Ranking

The first attempt was ranking paths according to capacity along the path. The shortest path was the path with the most capacity. This means that the weight of each edge e is $1/c_e$. Originally it was thought that this would be the best way to avoid congestion. However, after running some experiments, by generating models for several topologies and workloads using this path ranking technique and looking at the actual paths selected, as well as the solutions (or lack of) to the models, this method of sorting paths turned out to be not very good as latency-slow, roundabout paths were usually selected. Also using this method, subsequent paths were not as useful in solving the model as they were backing off from the path with the most capacity. Although subsequent paths may have been latency-faster, this was not actually considered when trying to find a solution.

To compare this against a standard latency-shortest paths method of routing, another model which keeps track of the number of undelivered messages was created. This model can use just one path per sender-receiver pair as in traditional overlay routing. Messages that cannot fit along a path are considered dropped. As the protocol can perform message splitting, a dropped message cannot simply be counted as one single undelivered message, as it may be split further along and be expected by many receivers. The model therefore counts the undelivered messages at receivers. This is usually greater than the number of sent messages. See Section 3.6.3 for the model description.

3.5.2 Latency Ranking (Bottleneck-Disjoint)

Results from the undelivered messages model for various topologies and workloads showed that when using a single latency-shortest path between senders and receivers, the number of undelivered messages was not significant. Therefore it was decided to change the path ranking technique to be based on latency, but to allow more than one path per sender-receiver pair as an attempt to avoid dropping messages altogether. This method ranks paths only by latency, which means that the algorithm does not consider capacity when selecting subsequent paths. A subsequent path selected this way may have more or less capacity than the previously selected path. However this method does select bottleneck-disjoint paths with respect to capacity.

The pseudocode for this algorithm is shown in Algorithm 1. As in [dQVMdS00], the graph is represented by the set of nodes $\mathcal{N}$ and the set of arcs $\mathcal{A}$ ($\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$). Let π_x for a node $x \in \mathcal{N}$ denote the shortest distance from the source node s to x. Let $d_{i,j}$ represent the distance between nodes i and j. Node n_i represents the i^{th} node n in a path p. Let $I(x) = \{(i,x)|(i,x) \in \mathcal{A}\}$ denote the set of incoming arcs to node x (the set of arcs whose head node is x). Let $T(x) = \{i \in \mathcal{N}|(i,x) \in I(x)\}$ denote the set of tail nodes for arcs in $I(x)$. Let $B(p)$ denote the set of bottleneck arcs in path p. Primes are used to mark nodes and $x^{(k)'}$ denotes node x with k primes. This means that $x^{(0)'} \equiv x, \forall x \in \mathcal{N}$. In the algorithm, $\overline{x}_h$ denotes the primed node x_h with no primes, $\hat{x}_{h-1}$ denotes the tail node of the arc of path p whose head node is x_h, and $\hat{x}_{h-1}$ can be a primed node as well as x_h. For finding the initial shortest paths tree, Dijkstra's algorithm was used.

Definition 3.5 (Bottleneck) *For a path p made up of links $L_p = \{l_1, ..., l_n\}$ where $l_i \in E, \forall i : 1, ..., n$, the bottleneck is the link l_i with minimum capacity c_{l_i}. Multiple bottlenecks may exist in a path, these are called $B_p = \{b_1, ..., b_m\}$ where $b_i \in L_p, \forall i : 1, ..., m$.*

```
begin
    determine a shortest paths tree of $(\mathcal{N}, \mathcal{A})$
    $p_1 \leftarrow$ shortest path from $s$ to $t$ in shortest paths tree
    $k \leftarrow 1$
    $p \leftarrow p_1$
    while there is an alternative to $p$ and $k < K$ do
        foreach $b \in B(p)$ do
            remove $b$ from $(\mathcal{N}, \mathcal{A})$
        end
        determine $n_h$, the first node of $p$ such that $\overline{n}_h$ has more than a single incoming arc
        if $n'_h \notin \mathcal{N}$ then
            add $n'_h$ to $(\mathcal{N}, \mathcal{A})$
            $T(\overline{n}_h) \leftarrow T(\overline{n}_h) - \{\hat{n}_{h-1}\}$
            $I(\overline{n}_h) \leftarrow I(\overline{n}_h) - \{(\hat{n}_{h-1}, \overline{n}_h)\}$
            $\pi_{n'_h} \leftarrow min\{\pi_x + d_{x,\overline{n}_h} | (x, \overline{n}_h) \in I(\overline{n}_h)\}$
            $n_i \leftarrow n_{h+1}$
        else
            $n_i \leftarrow$ the first node following $n_h \in p$ such that $n'_i \notin \mathcal{N}$
        end
        foreach $n_j \in \{n_i, ..., t^{(k-1)'}\}$ do
            add $n'_h$ to $(\mathcal{N}, \mathcal{A})$
            $T(\overline{n}_j) \leftarrow T(\overline{n}_j) - \{\hat{n}_{j-1}\} \cup \{n'_{j-1}\}$
            $I(\overline{n}_j) \leftarrow I(\overline{n}_j) - \{(\hat{n}_{j-1}, \overline{n}_j)\} \cup \{(n'_{j-1,\overline{n}_j})\}$
        end
        $p \leftarrow$ shortest path from $s$ to $t^{(k)'} \in (\mathcal{N}, \mathcal{A})$
        $B(p) \leftarrow$ bottleneck links in $p$
        $k \leftarrow k + 1$
    end
end
```

Algorithm 1: Calculation of K-shortest bottleneck-disjoint paths from sender s to receiver t (adapted from [dQVMdS00], but with added bottleneck disjoint path selection)

Definition 3.6 (Bottleneck-disjoint paths) *For a set of paths $p_1, ..., p_n$ from a sender $s \in N$ to a receiver $r \in N$ with bottleneck links $B_{p_1}, ..., B_{p_n}$ and bs_i bottleneck links in path i where $bs_i = |B_{p_i}|$, the paths are bottleneck-disjoint if $b_i \in B_{p_j}$, then $b_i \notin L_{p_k}$ where $\forall i : 1, ..., bs_j, \forall j : 1, ..., n, \forall k : j+1, ..., n$. This means that bottleneck-disjoint paths are disjoint with respect to all bottleneck links. Note that a bottleneck link from a future path may be a part of a previous path. For example when calculating the first path, bottlenecks from successor paths may be a part of this path, as it is not known in advance what they will be. However in this case, they will not be bottlenecks in the first path, because if they were they would not be a part of the subsequent* *paths.*

Having many paths available for routing is not very important if these paths have a large overlap. Subsequent paths will not be very useful towards finding a solution to the model. In the initial testing stages (ranking paths according to capacity) subsequent paths would only differ by a few nodes at most. After this, a disjoint path selection technique was used and results using this showed that making the paths disjoint reduces the number of required paths to route messages as more capacity is available.

An improvement to this is to make sure the paths are bottleneck-disjoint, not just as disjoint as possible. This idea came from the work presented in [NZ01]. Having completely disjoint paths means that each subsequent path is further and further away from the fastest path, i.e., slower. Bottleneck-disjoint paths allow for increased capacity with fewer paths while still maintaining speed along these paths. When a path is found, the bottleneck is determined and this edge is removed from the graph. Subsequent paths cannot contain this link.

3.5.3 Latency Ranking (Path Rejecting)

There were experiments with a modification to this path selection method which also ranks paths by latency, but only accepts subsequent paths if their bottleneck link has more capacity. This means that if a bottleneck link with less capacity is encountered in a candidate path, this path will be rejected and the next candidate path will be determined. This method does not remove any edges from the graph, but it has stricter requirements of the candidate paths. In other words the subsequent paths can only improve the odds of a solution being found to the model.

Models for various topologies and message workloads were generated using this path rejecting technique, but ranking paths using this path selection technique actually turned out to be significantly slower than the bottleneck-disjoint approach. Many more paths had to be considered and rejected by the shortest-paths algorithm until acceptable paths were found. This approach did manage to find acceptable solutions with fewer delivery graphs, howewer there was only a slight difference when compared to the bottleneck-disjoint approach and these delivery graphs took much longer to calculate.

As the overall number of paths that the algorithm can look at is limited due to memory restrictions, in some cases where the bottleneck-disjoint paths method lead to a solution, this way of selecting paths leads to no solution. A combination of the two approaches which lead to similar results as the path-rejecting approach was also tried. For this reason, ranking paths simply by latency and choosing bottleneck-disjoint paths with respect to capacity leads to better performance as finding paths is significantly faster without losing much in terms of the required number of paths per message until a solution is found.

3.6 Model Description

The models used are now formally described. First, the optimal model and then the suboptimal models are looked at. Originally the approximated solution was to be compared against an optimal model. The optimal model considers all possible paths and all possible ways of splitting flows along those paths. The paths are chosen by the model solver. In a sense the optimal model starts with infinite k and has freedom to use any number of paths and splitting combinations for each flow. The optimal model was intractable and therefore a direct comparison against the suboptimal model could not be made. Still, the optimal model is described as it gives an accurate depiction of the ideal strived for with the suboptimal model.

The difference is that paths are not globally considered in the suboptimal model, but instead preselected by a shortest-path algorithm. The model can only determine ways to split flows across these predetermined paths. Path selection is not done by the model solver in this case. As only these local paths are considered, the optimal path selection and splitting might be missed by the suboptimal model leading to greater network usage when compared to the ideal routing.

The actual paths themselves are not a part of the model. What is important is which nodes are in each delivery graph. These sets of nodes are numbered for each flow. For each flow $f \in F$ there are $1, ..., n_f$ delivery graphs. For the set of messages M and nodes V the variables used in the model are as follows:

$x_{ijfm} \in \{0, 1\}$	$x_{ijfm} = 1$ when node i sends message $m \in \{1, ..., \sigma_f\}$ of flow f to node j
σ_f	the number of messages in flow f
β_v	the aggregate incoming capacity at node $v \in V$

3.6.1 Optimal Models

The optimal versions of these models are now described. It should be noted that there were several other optimal models before the one described below. These other versions had a few shortcuts in an attempt at tractability. However these shortcuts made the models a less accurate representation of the protocol, and despite the shortcuts finding a (feasible) solution, still took far too long. Therefore only the following optimal models are included, representing the ideal which strived for with the suboptimal models.

The first constraint is to ensure that all receivers receive all messages of all flows to which they are subscribed:

$$\sum_{m \in \{1,...,\sigma_f\}} \sum_{j \in V} x_{ijfm} \geq \sigma_f \, |j|$$

$$\forall i \in V, j \text{ is subscribed to } f,$$

$$\forall f \in F$$

Here, $|j|$ represents the number of subscribers j subscribed to flow f.

Nodes can only send messages of flows if they receive those messages or if they create them:

$$x_{jkfm} \leq \sum_{\substack{i \in V \\ i \neq j}} x_{ijfm}$$

$$\forall j, k \in V, j \text{ is not publisher of } f,$$

$$\forall f \in F, \forall m \in \{1, ..., \sigma_f\}$$

This is necessary in order to prevent nodes from sending messages which they do not have.

The amount of flow sent to a node cannot exceed it's incoming aggregate capacity:

$$\sum_{\substack{i \in V \\ i \neq j}} \sum_{f \in F} \sum_{m \in \{1,\ldots,\sigma_f\}} x_{ijfm} \leq \beta_j$$

This constraint says that for a node j, the sum of the sizes of all of the flows which go through it cannot it exceed the given space it has β_j.

The goal is to minimise usage:

$$\sum_{\substack{i,j \in V \\ i \neq j}} \sum_{f \in F} \sum_{m \in \{1,\ldots,\sigma_f\}} x_{ijfm}$$

This is the summation of all of the x_{ijfm} variables, or all of the message traffic which is required for all receivers to receive the messages they want to receive.

In the case of the load-balancing model there is an additional constraint:

$$\sum_{\substack{i \in V \\ i \neq j}} \sum_{f \in F} \sum_{m \in \{1,\ldots,\sigma_f\}} x_{ijfm} \leq y$$

The goal in this model is simply to minimize y. By expressing the goal indirectly using y and together with this constraint, the model solver is forced to spread the traffic load as evenly as possible across the given network bandwidth.

In the next section, the suboptimal models which try to approximate the optimal models are discussed.

3.6.2 Suboptimal Models

The suboptimal versions of these models are now described. Along with the constraints, examples from actual generated models are provided to illustrate how the constraints are converted into a linear programming model which the solver can solve. These models are quite large and therefore only a small portion is shown here. Note that in the generated models constraints can only be expressed as inequalities, and variables can only be on one side of the inequality. Also, the first variable in the constraints has an extra plus sign before it, but this is simply to allow the model generator code to be cleaner and does not change the result.

The code for the model generator is show in Appendix A. For brevity, this does not include the path-finding code, but the pseudocode for this is discussed in Section 3.5.2. This code reads in files containing the topology, message workload and some other configuration parameters such as the seed to use for random publisher placement and network bandwidth capacities. This is explained in more detail in the next chapter.

The new variables introduced in the suboptimal models are listed below. Note that the variable x_{ijfm} has been replaced with x_{fi} as described below. This new variable is no longer binary; it is discrete and can have any value from 0 up to σ_f. The variables σ_f and β_v are still used to mean the same as they did in the optimal model.

x_{fi}	the amount of flow $f \in F$ sent along delivery graph i
γ_i	the number of nodes in delivery graph i

The values of x_{fi} are restricted based on the aggregate incoming capacity of the nodes. Typically a node will have multiple message flows going through it as it will be part of multiple delivery graphs. The total number of messages sent through a node cannot exceed its aggregate incoming capacity, therefore the amount of flow f sent along delivery graph i is restricted according to the following constraints:

$$\sum_{f \in F} \sum_{\substack{i=1,\ldots,n_f \\ v \text{ is part of delivery graph } i}} x_{fi} \leq \beta_v \quad \forall v \in V$$

These constraints prevent nodes from receiving too many packets and dropping them. An example from the generated model for 300 nodes and 10000 flows is shown below. In this case the link capacities have been set to 40 MBits and the aggregate link capacity at the node to which this constraint corresponds is 240 MBits. There are 300 of these type of constraints in this generated model, one for each node.

```
+ x_1_1 + x_1_2 + x_1_4 + x_1_5 + x_4_2 + x_4_4 + x_6_5 + x_7_1 + x_7_2 +
x_7_4 + x_7_5 + x_8_1 + x_8_2 + x_8_3 + ··· + x_9933_5 + x_9944_1 +
x_9962_1 + x_9968_1 + x_9973_1 + x_9976_1 + x_9982_1 + x_9989_1 <= 240
```

Also the amount of a message flow sent cannot contain more messages than are in that flow. This is why x_{fi} is restricted as follows:

$$x_{fi} \geq 0 \quad \forall f \in F, i = 1, ..., n_f$$

$$\sum_{i=1,\ldots,n_f} x_{fi} = \sigma_f \quad \forall f \in F$$

The bounds say that all of the messages in a flow must be sent and that no more of a flow may be sent along a delivery graph than the number of messages in that flow. In the generated model these two constraints are merged into one. Some of these constraints from the generated model for 300 nodes and 10000 flows are shown below. In this case there is one message per flow and $k = 5$.

```
+ x_1_1 + x_1_2 + x_1_3 + x_1_4 + x_1_5 >= 1
+ x_2_1 + x_2_2 + x_2_3 + x_2_4 + x_2_5 >= 1
+ x_3_1 + x_3_2 + x_3_3 + x_3_4 + x_3_5 >= 1
⋮
+ x_9998_1 + x_9998_2 + x_9998_3 + x_9998_4 + x_9998_5 >= 1
+ x_9999_1 + x_9999_2 + x_9999_3 + x_9999_4 + x_9999_5 >= 1
+ x_10000_1 + x_10000_2 + x_10000_3 + x_10000_4 + x_10000_5 >= 1
```

Given these constraints and bounds, two different goals are optimised for. One of the minimisation goals is to minimise network usage:

$$\sum_{f\in F}\sum_{i=1,\ldots,n_f}\sigma_f\gamma_i x_{fi}$$

This minimisation expression will seek to miminise the overall traffic sent over the entire network. A portion of the minimisation goal from the generated model for 100 nodes and 5000 flows is shown below. The constant $\sigma_f\gamma_i$ is precomputed by the model generator and then output as a single number before the x_{fi} variable. Note that a flow may not have a delivery graph for every k as can be seen here in the case of flow number 5000. Here, flow number 5000 has only a single delivery graph (when $k = 1$ in this case). In this case γ_i will be 0.

```
Minimize + 24 x_1_1 + 29 x_1_2 + 25 x_1_3 + 23 x_1_4 + 29 x_1_5 + 22 x_1_6 +
18 x_1_7 + 15 x_1_8 + 16 x_1_9 + 11 x_1_10 + 13 x_2_1 + 9 x_2_2 + 10 x_2_3 +
5 x_2_4 + 6 x_2_5 + 8 x_2_6 + 9 x_2_7 + 8 x_2_8 + 10 x_2_9 + 10 x_2_10 + ⋯ +
3 x_5000_1 + 0 x_5000_2 + 0 x_5000_3 + 0 x_5000_4 + 0  x_5000_5 + 0 x_5000_6 +
0 x_5000_7 + 0 x_5000_8 + 0 x_5000_9 + 0 x_5000_10
```

The other goal is to balance network usage. For this goal, an additional constraint is added to the model:

$$\sum_{f\in F}\sum_{\substack{i=1,\ldots,n_f \\ v\text{ is part of delivery graph } i}} x_{fi}\le y\quad\forall v\in V$$

An example from the generated model for 100 nodes and 5000 flows is shown below. For compactness, the whole constraint is not presented. In this generated model there are 100 of these type of constraints, one for each node.

```
+ x_1_2 + x_20_2 + x_23_1 + x_44_1 + x_183_1 + x_191_3 + x_201_1 + x_202_1 +
x_344_1 + x_353_1 + x_362_3 + ··· + x_4758_2 + x_4760_1 + x_4772_1 + x_4790_1 +
x_4803_2 + x_4808_2 + x_4928_1 + x_4970_3 - y <= 0
```

Then the goal is simply to minimize y. This means minimising the maximum number of messages sent to any one node.

These models are generated starting with $k = 1$. If one delivery graph per flow does not give enough capacity, k is incremented and a new model is generated until a solution can be found. Once a solution is found it describes how much of each flow to send down each delivery graph. The forwarding tables can be constructed from this information.

Note that flows refer to groups of messages from the same sender to the same set of receivers. This concept exists only in the model and is used to determine how to distribute a group of messages across its delivery graphs. Once a solution to the model has been found (i.e., the path each message will take has been determined), each message is treated individually, without regard to the flow that it belongs to. Therefore, after this chapter the concept of flows is ignored and the focus is on individual messages.

3.6.3 Undelivered Messages Model

The model used to count the number of undelivered messages in a workload is now discussed. This was used to model a standard latency-shortest path approach to routing, in order to see how many messages would be dropped. Therefore, multiple delivery graphs per flow are not allowed. However, this model is not allowed to use message splitting for a more fair comparison. Messages that do not fit in a delivery graph are considered to be dropped. As the delivery graphs do not need to be numbered for each flow, G_f is used to mean the delivery graph of flow f. The variables σ_f and β_v mean the same as they did in the suboptimal model described previously.

However, the variable x_{fi} from the suboptimal model now has a different meaning. It is still discrete and can have any value from 0 up to σ_f, but it now represents the amount of flow f that reaches node i. The set R_f is used to represent the nodes which are receivers of flow f. In this model, a new variable, d_{fr} is used to represent the number of undelivered messages of flow f at receiver $r \in R$. Like x_{fi}, this variable is discrete and can have any value from 0 up to σ_f. The constraints of the model are now described.

The first constraint says that no more of a flow may be sent to a node than the number of messages in that flow.

$$0 \leq x_{fi} \leq \sigma_f \quad \forall f \in F$$

Next, a node is allowed to receive only up to as much of a flow as its predecessor in the delivery graph has of that flow.

$$x_{fj} \leq x_{fi} \quad \forall f \in F, \forall (i,j) \in G_f$$

A node can only receive as much of a flow as it has room for.

$$\sum_{f \in F} x_{fi} \leq \beta_i \quad \forall i \in V$$

The last constraint states that the number of undelivered messages of a flow at a receiver is the number of messages in that flow minus the received messages of that flow.

$$d_{fr} \geq \sigma_f - x_{fr} \quad \forall f \in F, \forall r \in R_f$$

The goal in this model is to minimize the number of undelivered messages at the receivers.

$$\sum_{f \in F} \sum_{r \in R_f} d_{fr}$$

Note that dropped messages cannot simply be counted because message splitting has to be taken into account. If a message is lost en route to a receiver and it is counted as one dropped message, this does not take into account message splitting which may occur further along the path to the receiver. Therefore the number of lost messages needs to be counted at the receivers.

3.7 Summary

In this chapter, the stage was set for the routing problem to be solved and the approach to solving it was described. The approach of casting it as an optimisation problem was discussed and terms used to talk about this problem and in describing its solution were defined. What it means to solve this problem and what is expected from the solution was also discussed. The different shortest-path ranking approaches used in creating delivery graphs were explained. Finally the different models used in CAMPR were defined. In the next chapter, some demonstrations of the models used in CAMPR are looked at.

Chapter 4

A Generative Demonstration of the Models

4.1 Introduction

In this chapter, a demonstration of the models discussed in the previous chapter is presented. As discussed in Section 3.6, the optimal models presented in the last chapter are intractable and therefore the suboptimal models were used to show the behaviour of CAMPR. The process of carrying out the demonstrations as well as the environment used and the various parameters explored during the demonstration are described. Graphs showing the results from the models are presented and analysed. The claim to be shown through the demonstration is now discussed.

It is typical in overlay networks to route traffic using a single path between a sender and a receiver. Referring back to the scenario described in Section 1.1 (Figure 1.1), it can be seen that this results in inefficient use of network resources. Packets are dropped and have to be retransmitted. Retransmissions may cause further drops, slowing down message delivery and wasting network resources which could have been put to better use.

To make the best use of network resources, message flows should be allowed to use more of the capacity available in the network so that time wasted retransmitting dropped packets can be avoided. Network resources should be used more efficiently by using path splitting and message splitting. These techniques are modelled and it is claimed that using a linear programming solver, a solution to this model can be found and used to create routing tables for a routing protocol which performs message splitting and path splitting. This would support more traffic than a traditional overlay routing approach using the same network resources.

In a traditional overlay routing approach, there is no path splitting. A single path is used to deliver each flow; there is no further selection of paths or avoiding of bottlenecks, etc. Paths are selected based on latency: the path with the least latency is the shortest path. However, when comparing, message splitting can occur in the traditional approach.

For comparison purposes, it is assumed that messages that cannot fit in the capacity along this path are dropped and retransmitted. Supporting a bigger workload means that for a given topology, more messages can be routed without any drops. To test this, the undelivered messages model presented in the last chapter was used. This was tested with the same message workloads used to test CAMPR.

A routing approach which supports bigger workloads has some drawbacks compared to the traditional overlay routing approach. One drawback is the time taken to find paths (to construct forwarding tables) which provide more capacity for a flow. In CAMPR, multiple bottleneck-disjoint paths are calculated between senders and receivers. This takes longer than the traditional single-path latency-shortest path approach.

How a flow can be split across these paths also has to be considered. On top of that, message splitting can occur as a way to save bandwidth. Finding a routing solution which takes advantage of these things takes longer than the simpler traditional single-path approach.

Of course, a more complicated problem is solved, so it takes longer because more computational power is required. Note that in a traditional overlay routing approach the time lost waiting for message retransmissions is lost at the endpoints, by the receivers themselves. This can vary by the size of the flow to which a receiver is subscribed, the position of the receiver in the network and many other factors. In general, it can vary wildly by receiver.

In this approach, time is taken to calculate the forwarding tables but all receivers wait an equal amount of time for this to happen. Also in an actual implementation, the computation of the forwarding tables would not happen at the endpoints. So the extra computational power required at the receivers does not differ from the traditional approach.

4.2 Demonstration Process

To determine if the goals were achievable, a framework through which models could be generated with different parameters was created. This was presented in Figure 3.1. The main idea behind the framework is to allow the accurate recreation of specific network configurations, so that the effectiveness of CAMPR can be determined without actually implementing a prototype and deploying it on a network testbed. The framework for generating forwarding tables is now described.

First, the BRITE [MLMB01] topology generator is used to generate a topology. BRITE assigns latency to links, but capacities are also assigned to the links in this work. Latency is used as the weight of the edges, so the shortest path between two nodes is the path with the least latency. The model uses aggregate link capacity at a node to determine whether a message can be sent to it or not. For example if a node has three incoming links, each with capacity 10 Mbps, then the space available for receiving messages at this node becomes 30. This means that 30 messages can be sent to this node. The capacity values presented in Table 4.1 were used.

At first, the Siena Synthetic Benchmark Generator (SSBG) was used to generate publish/subscribe workloads. These workloads are sets of messages and subscriptions generated according to a specific distribution and attribute space for the number of nodes in the topology. The attribute space is the number of words which exist in the workload from which subscriptions and messages can be formed. The subscription size of nodes can also be set. The subscription size in conjunction with the attribute space determine the popularity of messages. For example a small attribute space with a large subscription size will lead to most messages being received by most receivers. A large attribute space with a small subscription size will lead to the opposite.

This tool was used to evaluate the Siena Fast Forwarding (SFF) algorithm [CW03]. As the SFF algorithm is meant to be used for forwarding in content-based networks, SSBG produces content-based workloads. At first, SSBG was modified to produce simpler workloads, which could be fed in to SFF in order to determine which messages get routed to which subscribers.

However, using SSBG in conjunction with SFF did not allow for precise tweaking of some parameters in the workloads, so a separate workload generator was written. Using this generator, the average popularity of messages could be set without having to deal with attribute spaces and subscription sizes to achieve an approximate popularity as before. Also workload generation became a one-step process, there was no more matching of messages to subscriptions involved. The output was in the same format as that produced by SFF. This allowed the replacement of SSBG and SFF with the new workload generator.

4.3 Demonstration Parameters

The parameters used in the demonstration are now described. As it was not known in advance which parameters will lead to interesting results, a large parameter space was explored in an attempt to recreate various different network scenarios. Some of the parameters in Table 4.1 were chosen because they were used in [CRW09].

Message workloads which are lists of receivers per message were generated. For simplicity each message is the same size. The size used is 1, but the solution can be used as a ratio mapped to any size. Different message popularities were used to simulate light and heavy workloads.

Parameter	Range
Popularity (% receivers)	5, 25, 45, 65, 85
Workload size (number of messages)	1000, 2000, 3000, 4000, 5000, 10000
Link capacity (Mbits)	20, 30, 40, 50, 60
Topology size (number of nodes)	100, 200, 300, 400, 500, 1000

Table 4.1: Parameters used and the ranges of their values

The popularity of a workload is the average percent of receivers which are subscribed to its messages. For example the messages in a workload with 25% popularity will be subscribed to by 25% of the receivers in the topology.

In the topology, 75% of the nodes were receivers and 10% were senders as this is the approach taken in [CRW09]. A node can be both a sender and a receiver, or neither, serving only as a transit node. The number of receivers per message is uniformly distributed. Popularities of 5%, 25%, 45%, 65% and 85% were used as shown in Table 4.1. Realistically there would probably never be content that has 85% popularity in a network, but this is just to simulate a worst-case scenario.

The size of a workload is the number of messages in it. Various sizes of workloads were used: 1000, 2000, 3000, 4000, 5000 and 10000 messages, as shown in Table 4.1. This is the number of messages before any splitting occurs. In the workloads, 75% of the nodes act as receivers, so for example if the popularity of a 10000 message workload is 85% and there are 1000 nodes, then the number of total messages expected to be delivered is .75 * 1000 * .85 * 10000 = 6375000 messages. Note that the workloads do not have timing information stating when each message is published. As these workloads represent the worst case, all of the messages are sent at the same time.

Different link capacities are assigned to the topology. As shown in Table 4.1, 20, 30, 40, 50, and 60 Mbits are used for the capacities. Using a capacity of just 10 Mbits lead to no solution in almost all cases and therefore this case is ignored as the results are not interesting.

The BRITE topology generator was used to generate topologies using the default parameters. Topologies with 100, 200, 300, 400, 500 and 1000 nodes were used as show in Table 4.1.

Generating models for topologies beyond these sizes is possible, but it takes prohibitively long so 1000 nodes is the largest topology used.

A particular parameter set (popularity, workload size, link capacity and number of nodes) is ran ten times using ten different seeds. Ten runs were used because this produced results with a negligible variance. The seeds determine which nodes are the publishers of messages. Different placement of the publishers in the network may affect the solution. Ten runs are done to avoid having a solution where the publisher placement happened to be lucky, leading to a trivial solution. Depending on the placement of publishers, more paths may be required to find a feasible solution. This also means that the time taken to find a solution differs depending on the seed.

Each parameter set starts with one path (i.e., $k = 1$) per sender-receiver pair in each delivery graph to generate the model. This first path is the latency-shortest path found by the shortest-path algorithm used. If the model cannot be solved using this (because of there being not enough capacity using a single path), k is incremented to find more paths to add to the delivery graphs and generate a new model using the new delivery graphs. Consequent paths found will be bottleneck-disjoint with respect to capacity and will have a higher latency, but may contain more link capacity. This process is repeated until a solution is found, or until k reaches one hundred. At this point it is considered that there is no solution for that particular parameter set. This number was chosen in the interest of time, as almost all of the models can be solved with less than 10 delivery graphs ($k = 10$).

The models were run on a computer with an Intel Xeon X5650 CPU with 32GB of RAM. This CPU has 24 cores clocked at 2.67GHz. Although there was access to many cores, there was not much benefit from them in terms of speed. Although the model solver does scale to multiple cores, the code for finding paths between nodes is run only on a single core.

The path-finding accounts for about 99% of the runtime and therefore similar results would have been achieved using only a single-core machine. However, having many cores and lots of memory did allow for multiple instances of the model generator to be run in parallel. As many different parameters are explored, this saved a lot of time.

In the next sections, some of the graphs generated are shown. First, the traditional overlay routing approach showing undelivered messages is looked at, followed by CAMPR, showing the required number of paths to route the messages without dropping them.

4.4 Traditional Overlay Routing

In this section, some of the graphs which were generated using a traditional overlay routing approach are shown. As mentioned previously this means no path splitting; only a single path between sender-receiver pairs. In solving the models representing traditional overlay routing, $k = 1$ is the maximum, so that a single delivery graph per message flow is used.

The graphs show the percentage of undelivered messages for different popularity levels, link capacities, topology sizes and workload sizes. The graphs show the percent of undelivered messages as well as the absolute number of undelivered messages. The lines correspond to the percent shown on the left scale and the bars correspond to the absolute number shown on the right scale. The way these are counted at the receivers, instead of at the point where the message is dropped as this would not be the actual number of unreceived messages due to message splitting is discussed in Section 3.6.3.

In the traditional approach, the same message workloads that were used to test CAMPR are used. The same ten seeds to position the publishers within the network are also used. The only difference from the CAMPR models is that there is no path splitting in these models. However, for comparison purposes message splitting is allowed in this approach. The variance of the results is negligible and therefore not shown.

The models have been generated ten times with different seeds for each data point, but there might be a slight discrepancy going across topology sizes. The reason is that workloads for a certain topology with a certain popularity are not subsets of those for a larger topology with the same popularity. For example, a workload of 1000 messages for 100 nodes and 5% popularity might not be contained in the workload of 1000 messages for 200 nodes and 5% popularity. The reason for this is to be able to maintain the same distribution across workloads when increasing the number of nodes.

Going across the number of messages, the workloads are contained in the larger workloads. For example the workload of 1000 messages for 100 nodes and 5% popularity is completely contained in the workload of 2000 messages for 100 nodes and 5% popularity. Going across popularity levels the workloads are also contained. For example the workload of 1000 messages for 100 nodes and 5% popularity is contained in the workload of 1000 messages for 100 nodes and 25% popularity. In this case there may be a slight difference in the receivers of messages.

The larger workloads will have the receivers of multiple messages from smaller workloads mapped onto a single message. For example if in a smaller workload message 1 is received by nodes 1, 2 and 3 and message 2 is received by nodes 4, 5 and 6, in the larger workload it may be the case that message 1 is received by nodes 1, 2, 3, 4, 5 and 6 and message 2 is received by receivers of messages 3 and 4 in the smaller workload. However it may also be the case that message 1 is received by nodes 1, 2, 3, 4 and 5 and message 2 is received by node 6 and more of the receivers for the following messages in the smaller workload (those of message 3 and onwards).

What this means is that for example in the case of the 5% and 25% popularity workloads the receivers of every set of 5 messages from the 5% workload may not be cleanly mapped onto a single message in the 25% workload. The important thing is that the average desired popularity is maintained.

For a given topology, as the number of messages increases the percentage of undelivered messages increases as more load is placed on the network. In a topology of 100 nodes, going from 5000 messages (Figure 4.1) to 10000 messages (Figure 4.2), it can be seen that the number of undelivered messages increases for all capacity sizes and popularity rates.

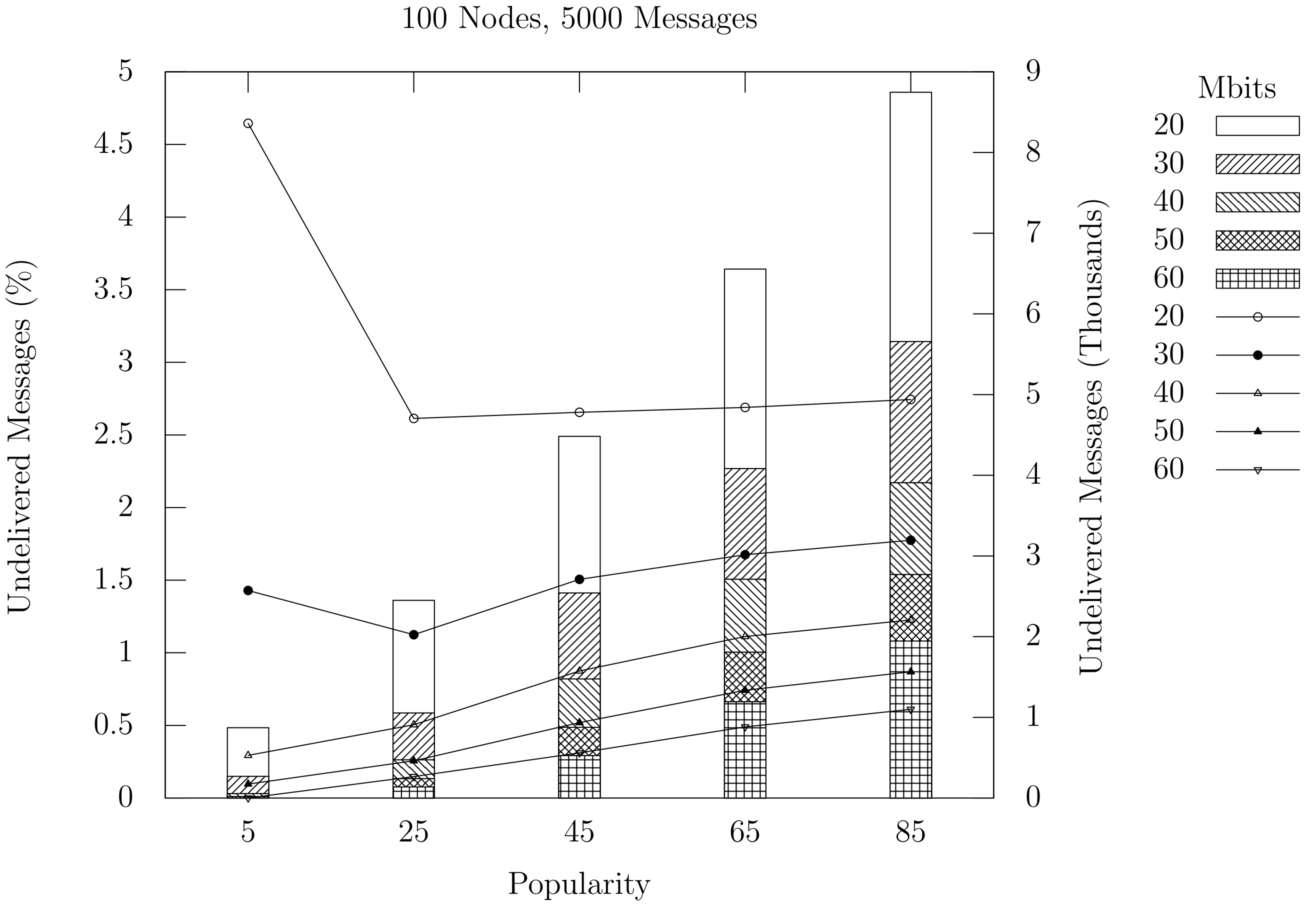

Figure 4.1: Undelivered messages for 100 nodes and 5000 messages

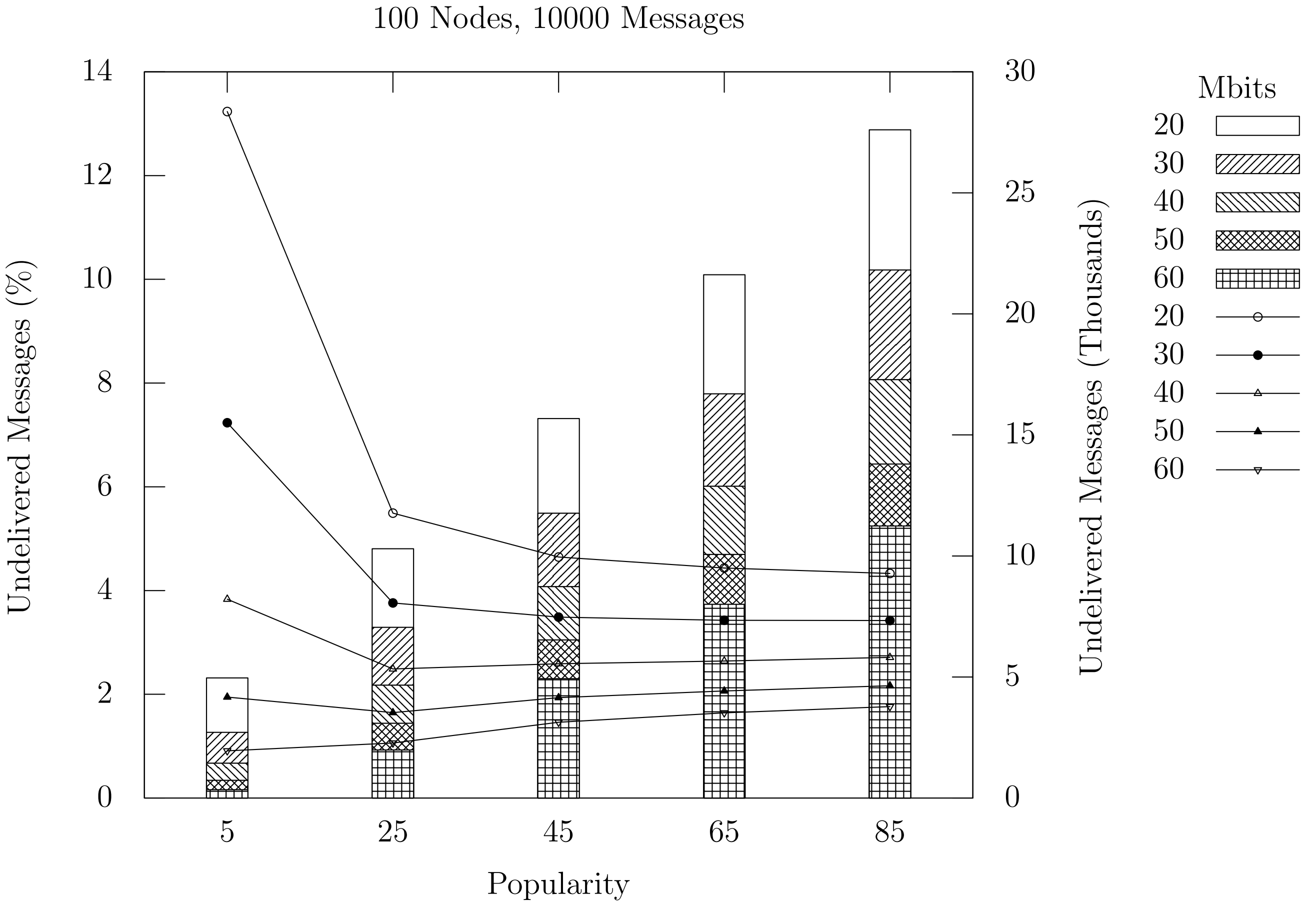

Figure 4.2: Undelivered messages for 100 nodes and 10000 messages

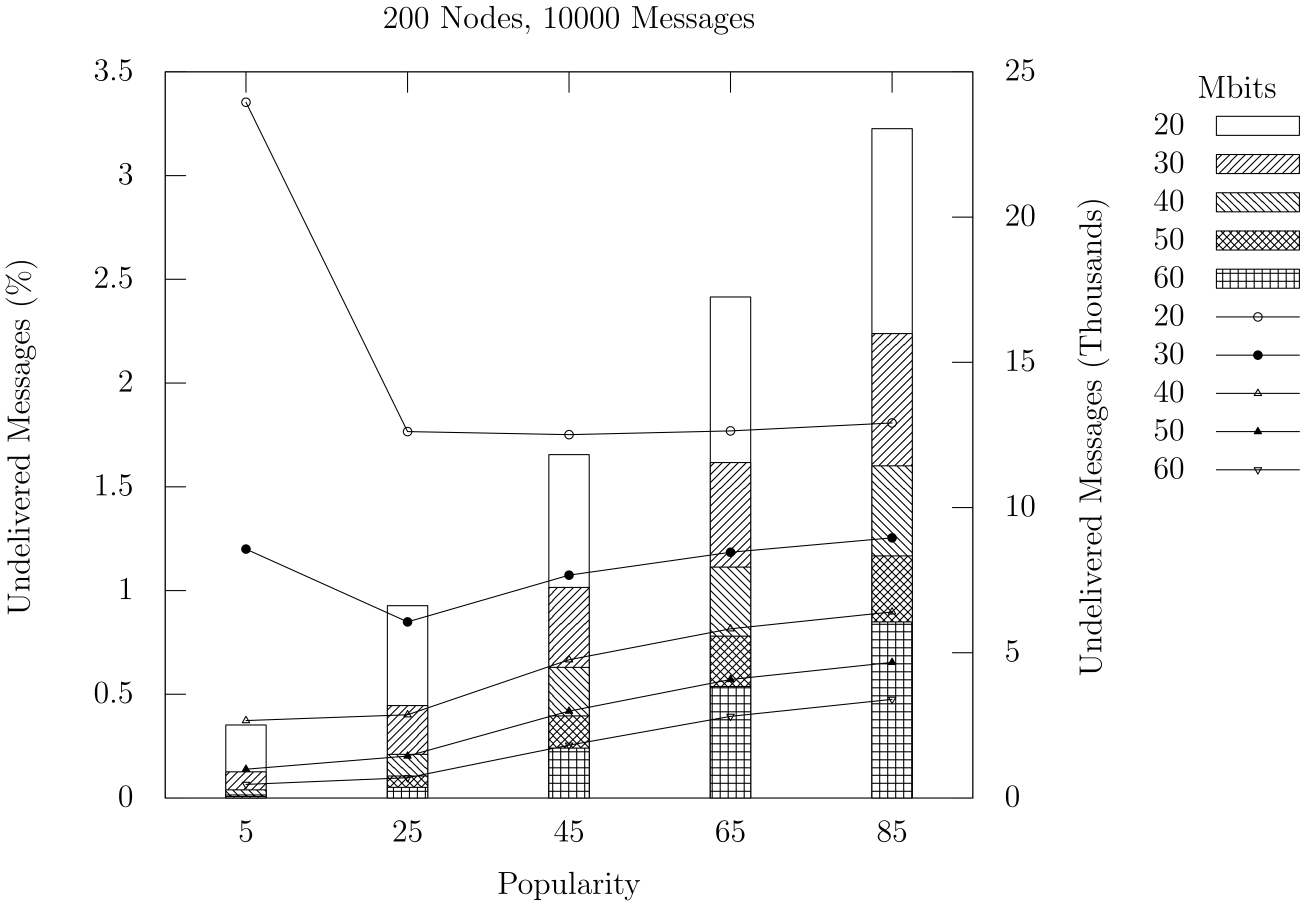

Figure 4.3: Undelivered messages for 200 nodes and 10000 messages

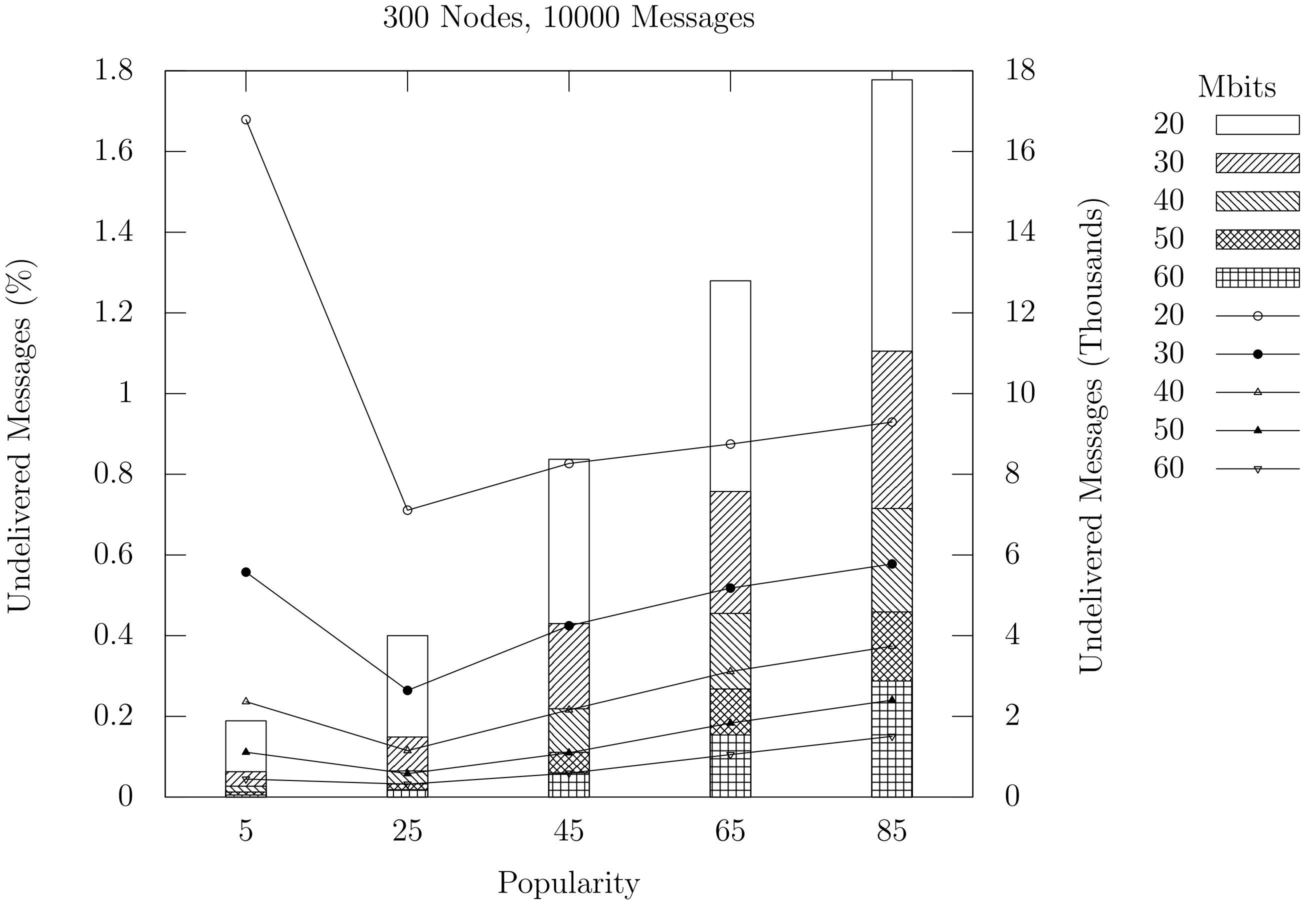

Figure 4.4: Undelivered messages for 300 nodes and 10000 messages

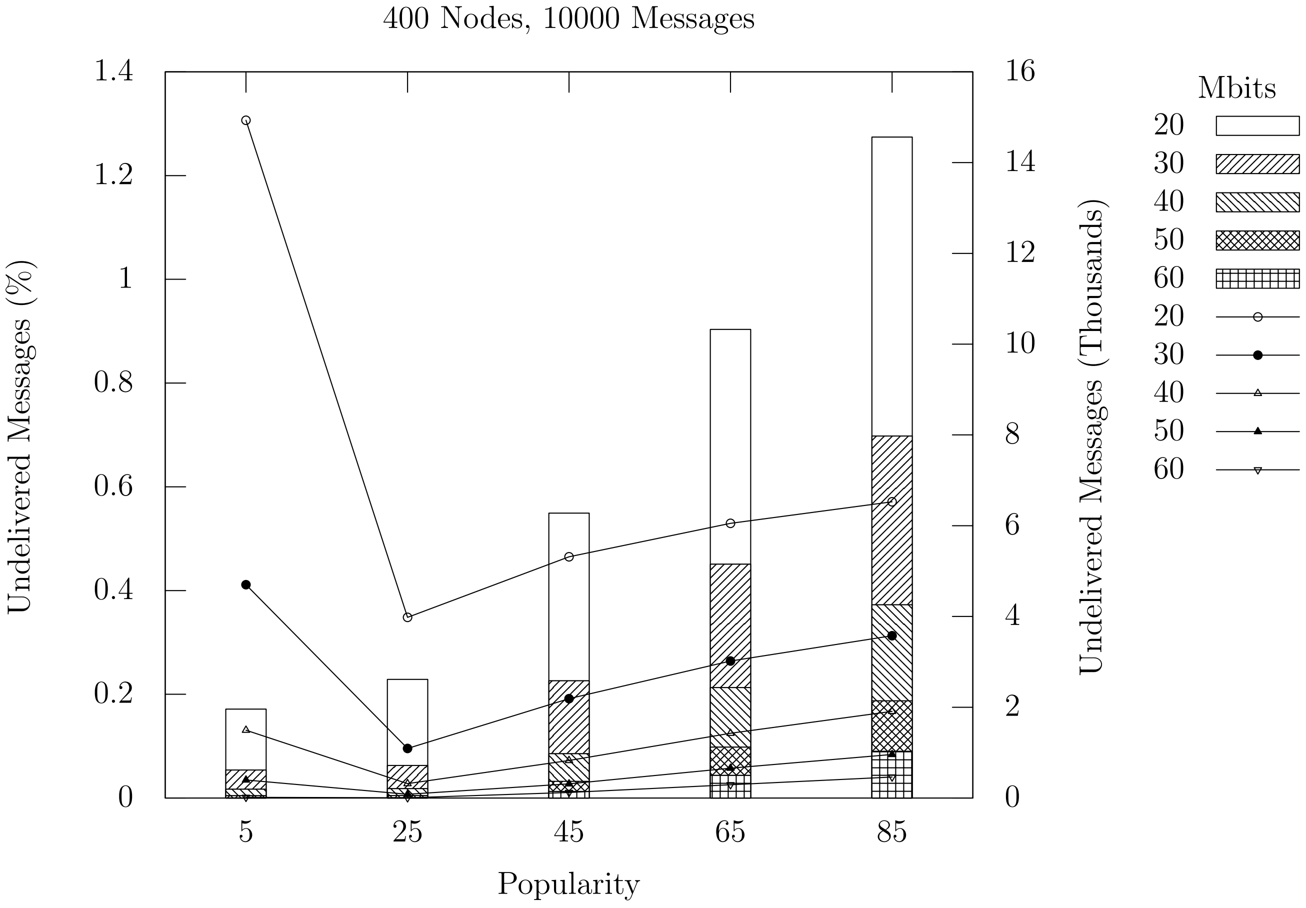

Figure 4.5: Undelivered messages for 400 nodes and 10000 messages

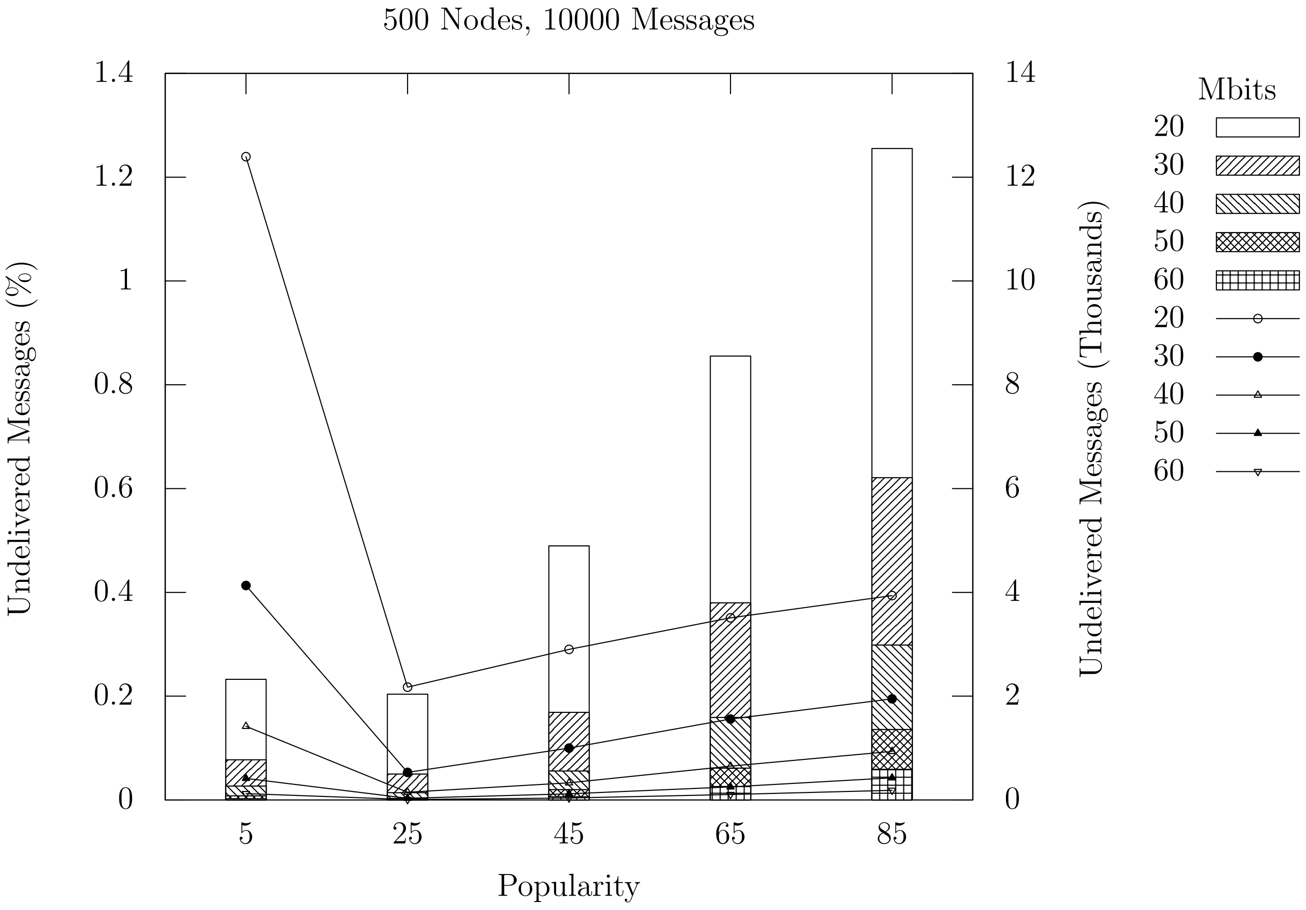

Figure 4.6: Undelivered messages for 500 nodes and 10000 messages

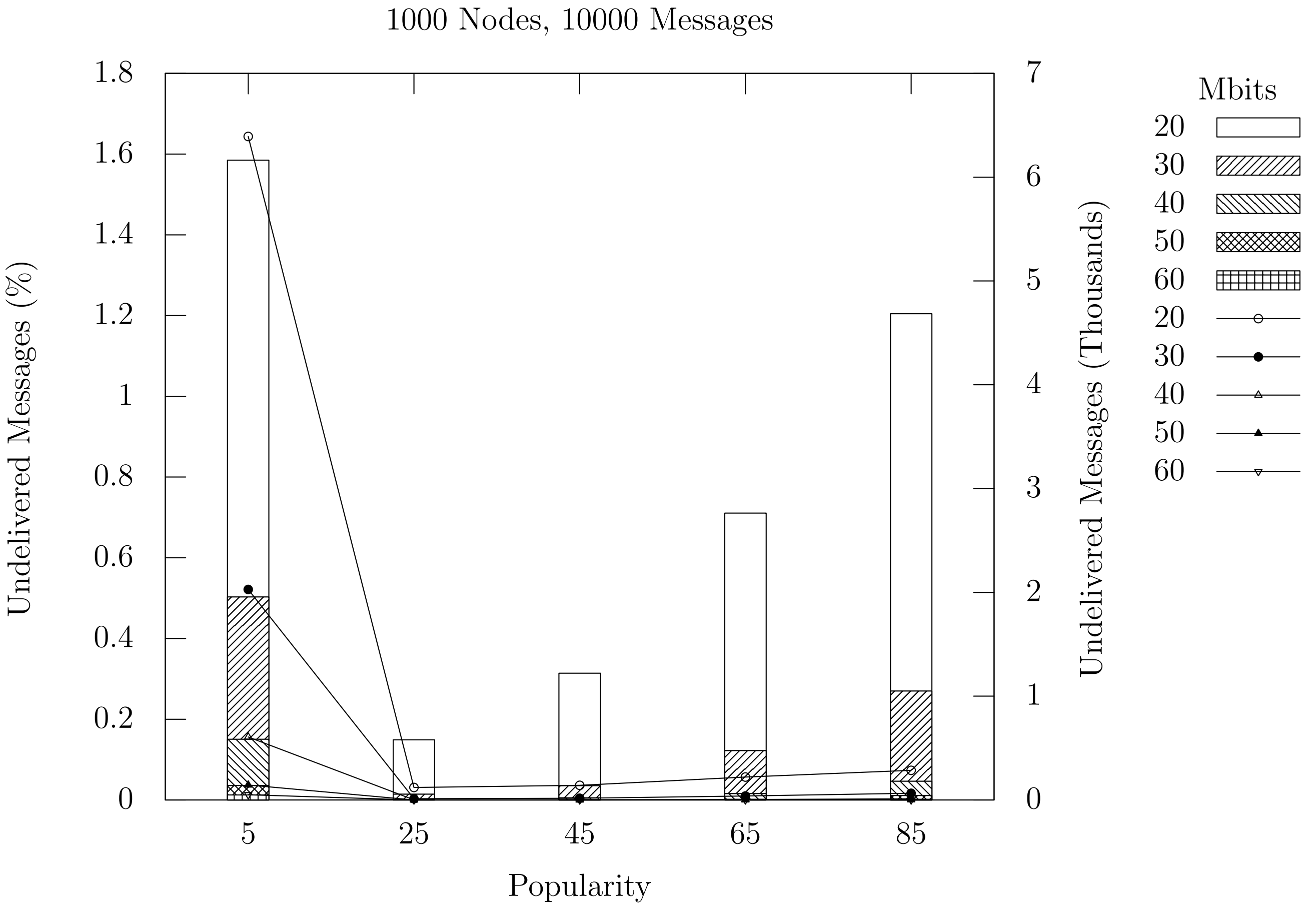

Figure 4.7: Undelivered messages for 1000 nodes and 10000 messages

For example in Figure 4.1, when the link capacity is 20 MBits It can be seen that for popularity 65% about 6500 messages are undelivered and for popularity 85% about 8800 messages are undelivered, but in Figure 4.2 the values for these parameters increase to about 22000 and 27500 messages respectively.

Within the same topology and the same message workload, when the link capacity is increased, the number of undelivered messages decreases. For example, in Figure 4.3 for popularity 85% it can be seen that for link capacity 20 MBits there are about 23000 undelivered messages (about 1.8%), but with link capacity 30 MBits, this decreases to about 16000 undelivered messages (about 1.25%).

Within the same topology and the same message workload, by increasing the popularity rate, the number of messages is increased, therefore increasing the number of undelivered messages. In Figure 4.4, for link capacity 30 MBits, at 45% popularity there are just over 4000 undelivered messages (about 0.4%), but at 65% popularity there are about 7500 undelivered messages (about 0.5%).

For a given message load, as the number of nodes increases the percentage of undelivered messages decreases as the algorithm has more paths to choose from along which to send the messages. Going from 400 nodes (Figure 4.5) to 500 nodes (Figure 4.6) to 1000 nodes (Figure 4.7) it can be seen that the number of undelivered messages drops.

Note that these graphs do not take into account time. The percentages of undelivered message shown is at one instant, in a “clean” network state. This is the best case. If the same workloads were to be run multiple times in quick succession, taking into account the “leftovers” from the previous run, there would be more undelivered messages as there would be less and less available network capacity for successive runs. In the traditional approach this buildup of undelivered messages gets worse over time as more and more resources are wasted on dropped messages.

4.5 CAMPR

In this section, some of the graphs which were generated using CAMPR are shown. This includes path-splitting and message splitting. The model generator is allowed to run until k reaches 100 and if at this point a solution has still not been found, this set of parameters is considered to have no solution and the next set of parameters are used. A feasible solution means that the messages fit within the capacity of the delivery graphs. The graphs show the number of required delivery graphs (the value of k) until a feasible routing solution is found.

Relating back to the graphs of the previous section, this means routing without overloading the network capacity and therefore not having undelivered messages. Graphs showing the time taken until a solution is found for the corresponding parameters are also included. The variance of the results is negligible and therefore not shown.

4.5.1 Required k

The graphs showing the required k to find a solution have five bars per popularity value. These bars represents the different values in MBits used for link capacities: 20, 30, 40, 50 and 60 respectively.

If the link capacity is constant (i.e., looking at just one color across different popularity values within a graph, for example in Figure 4.8), the required k does not change much. When the popularity of messages increases, each message needs to be sent to more subscribers, but because of message-splitting this does not increase the number of paths required by much.

If the popularity of messages is constant (i.e., looking at just one popularity value across different link capacity values within a graph, for example in Figure 4.9), it can be seen that the required k decreases as link capacity increase. This is also expected as messages need to be split over fewer delivery graphs when each graph can hold more data.

If the number of nodes is constant, when looking at different numbers of messages sent (Figures

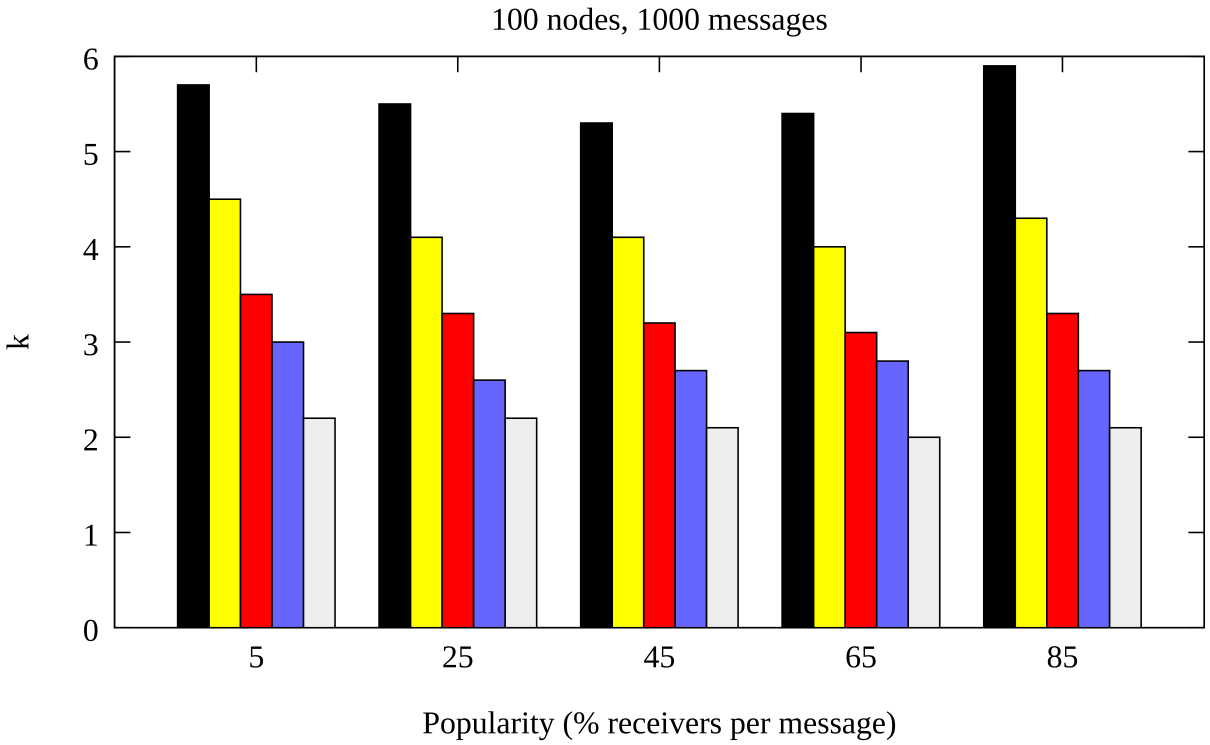

Figure 4.8: The required k to find a solution for 100 nodes and 1000 messages

4.10 and 4.11), it can be seen that the required k does not increase very much. This means that CAMPR can support large message workloads without requiring many more delivery graphs to be calculated. Of course, the number does increase as there is simply more data in the network.

If the number of messages is constant, when looking at different topology sizes (Figures 4.12, 4.13, 4.14 and 4.15) it can be seen that the required k decreases as topologies become larger. This is expected as the algorithm has more available paths along which it can choose to send messages.

Overall the graphs show that a feasible solution can be found with a few k for large workloads. Keep in mind that the listed message numbers are the messages published, before any splitting is done on them. For example in the case of 200 nodes, 4000 messages and 45% popularity, the total number of messages to be delivered is $.75 * 200 * .45 * 4000 = 270000$ messages to be delivered. The .75 is the percentage of nodes that are receivers. In this case 45% will receive the messages. For this topology and workload size, a solution can be found with about 5 or

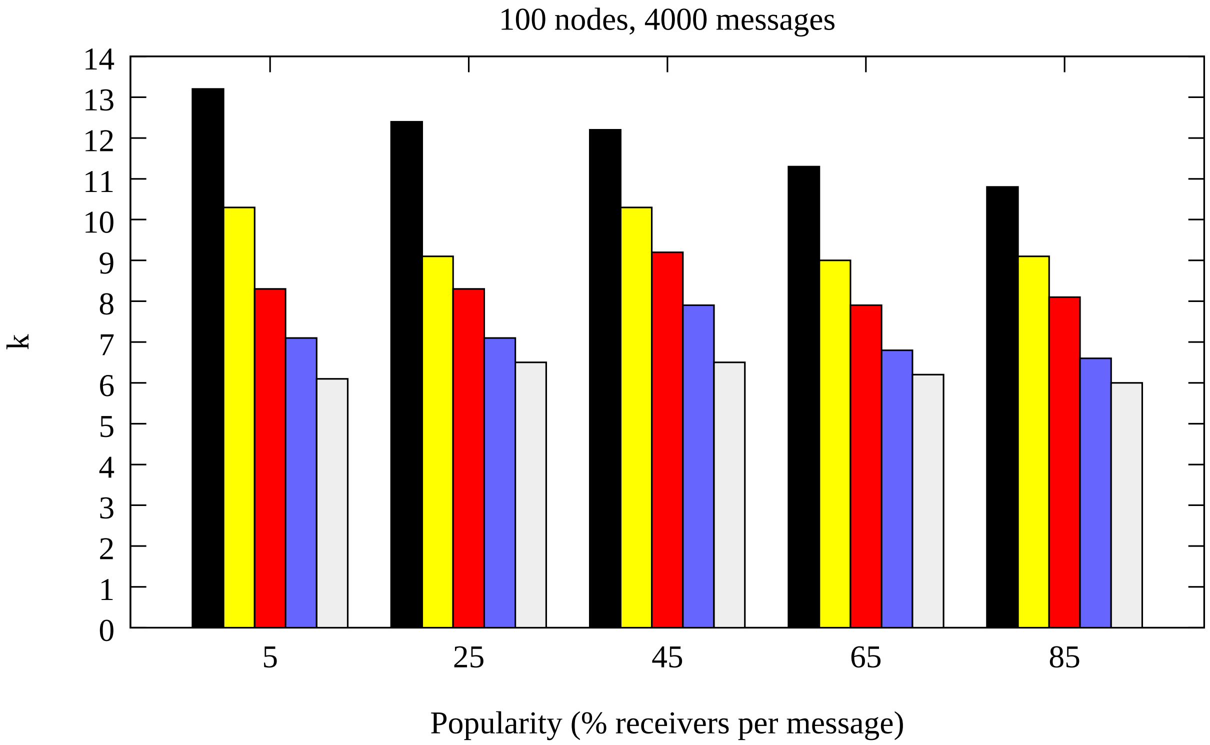

Figure 4.9: The required k to find a solution for 100 nodes and 4000 messages

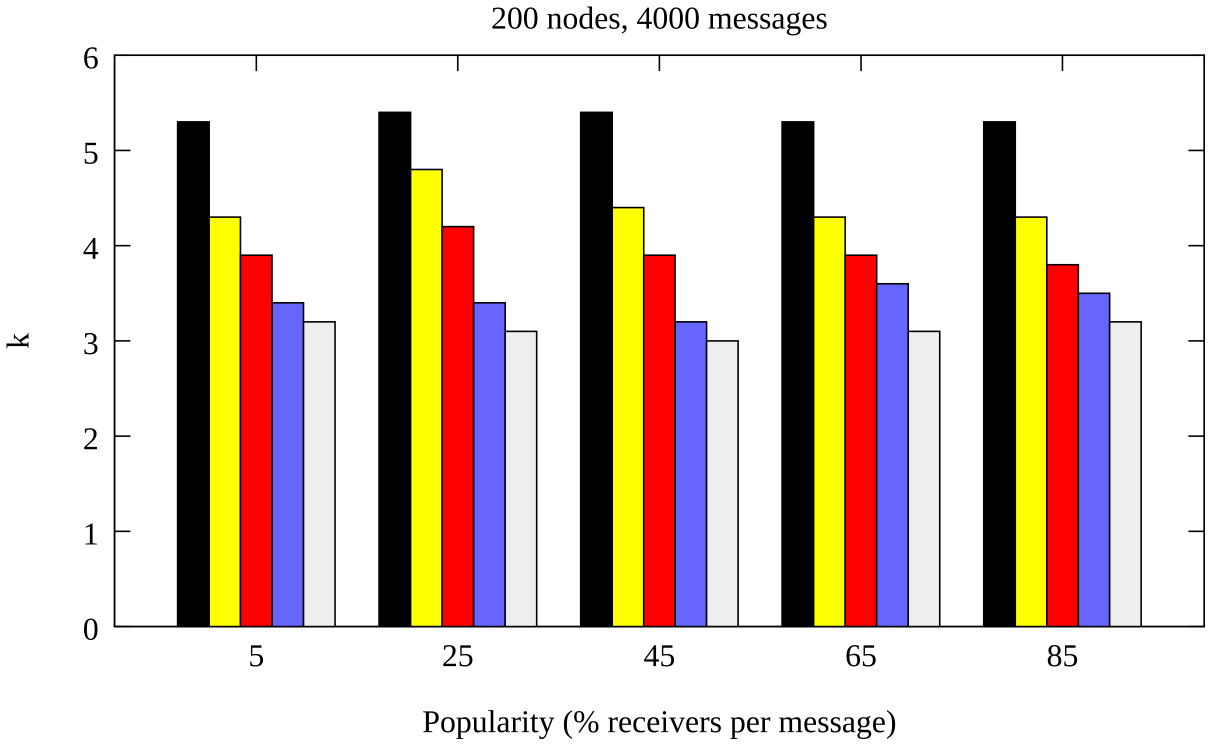

Figure 4.10: The required k to find a solution for 200 nodes and 4000 messages

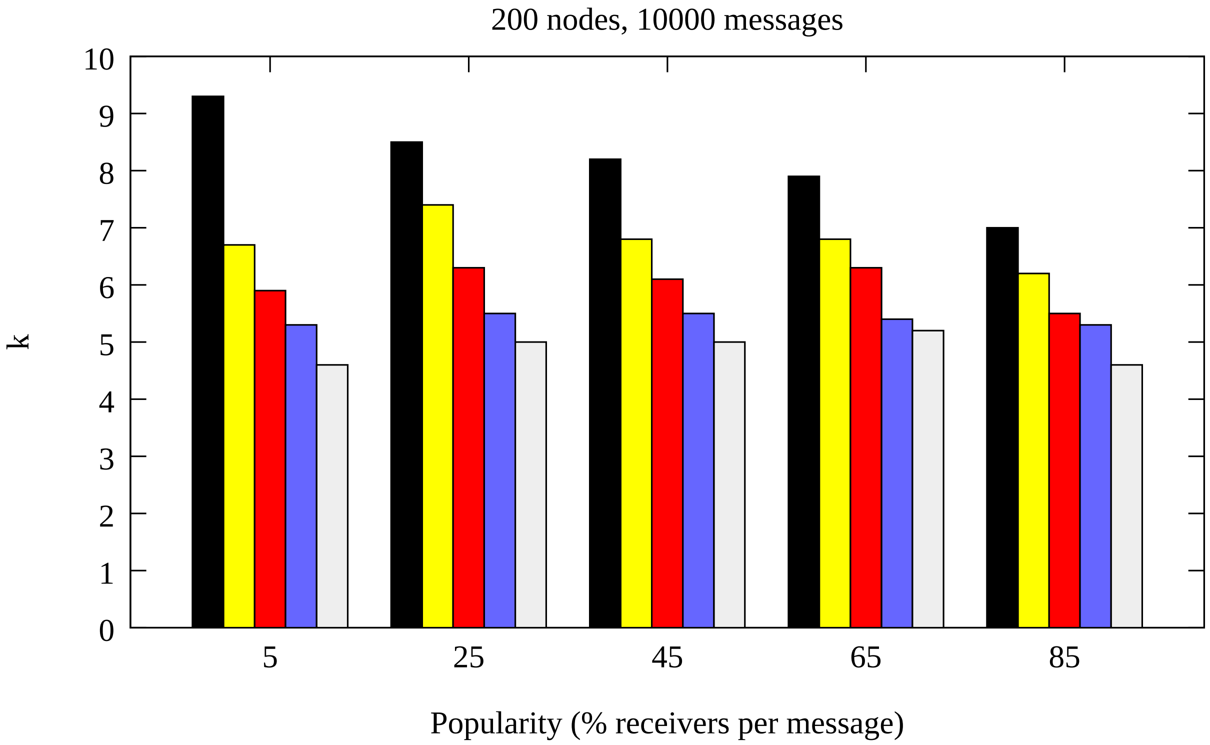

Figure 4.11: The required k to find a solution for 200 nodes and 10000 messages

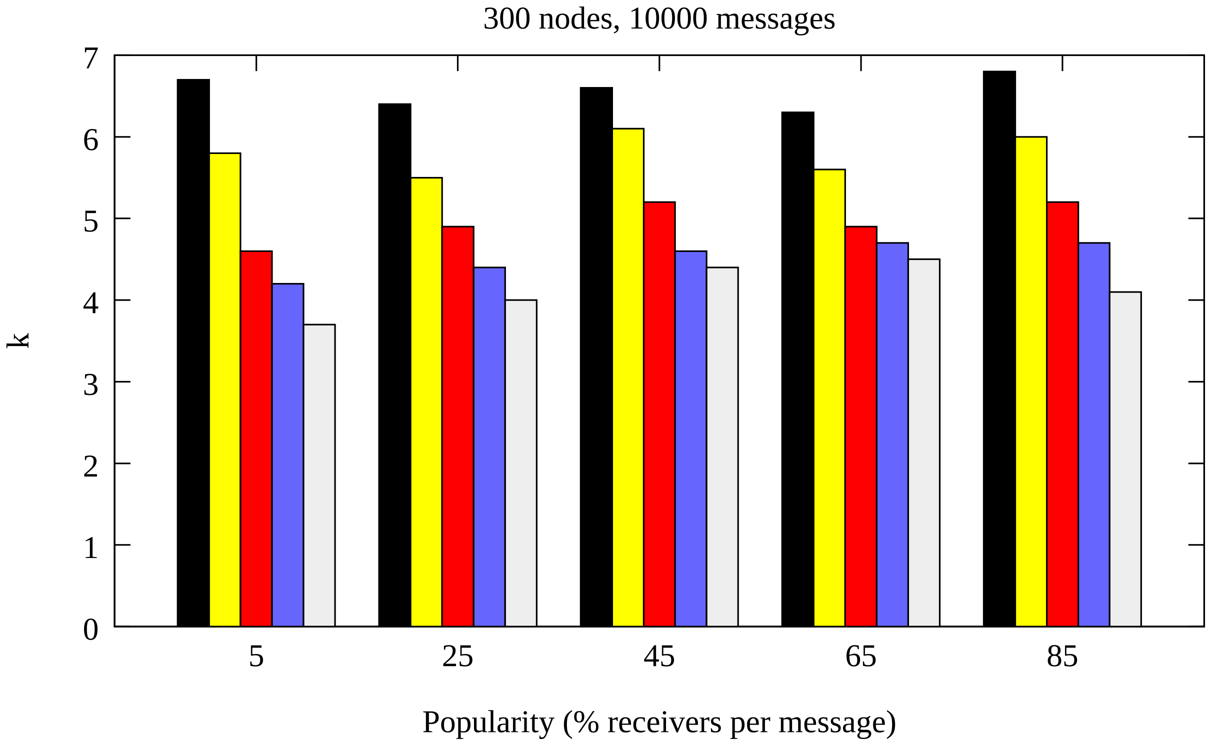

Figure 4.12: The required k to find a solution for 300 nodes and 10000 messages

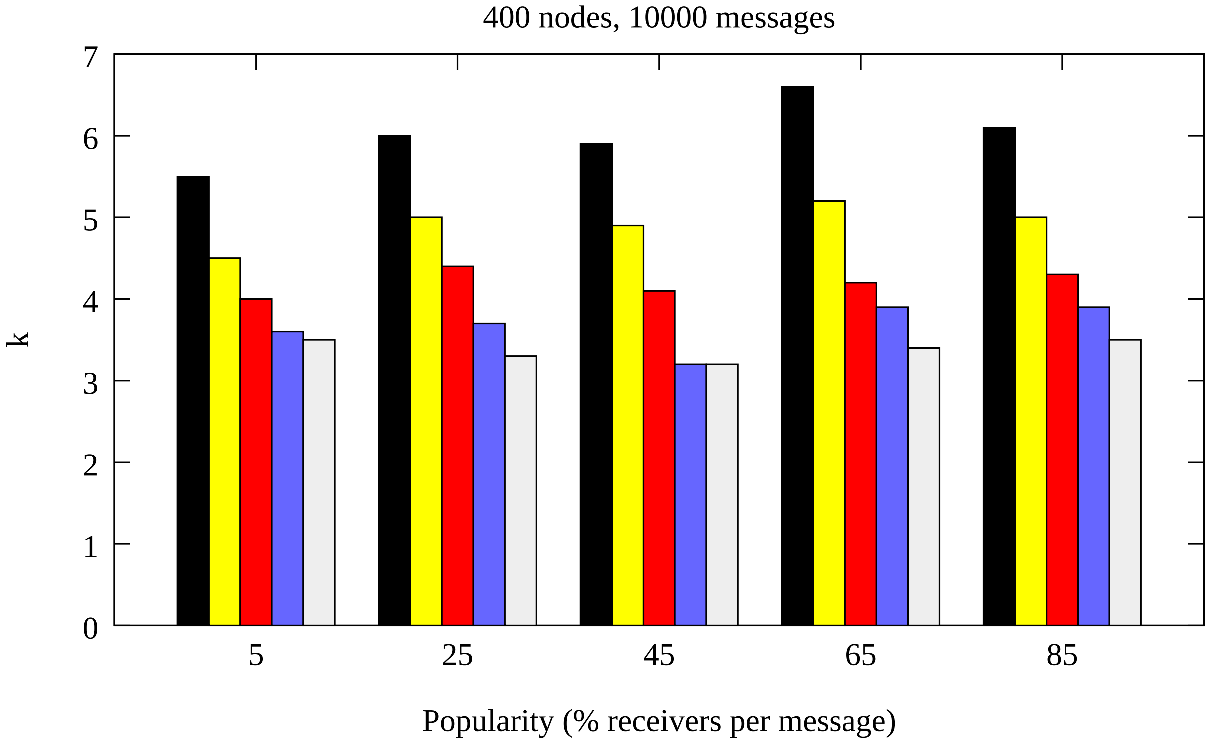

Figure 4.13: The required k to find a solution for 400 nodes and 10000 messages

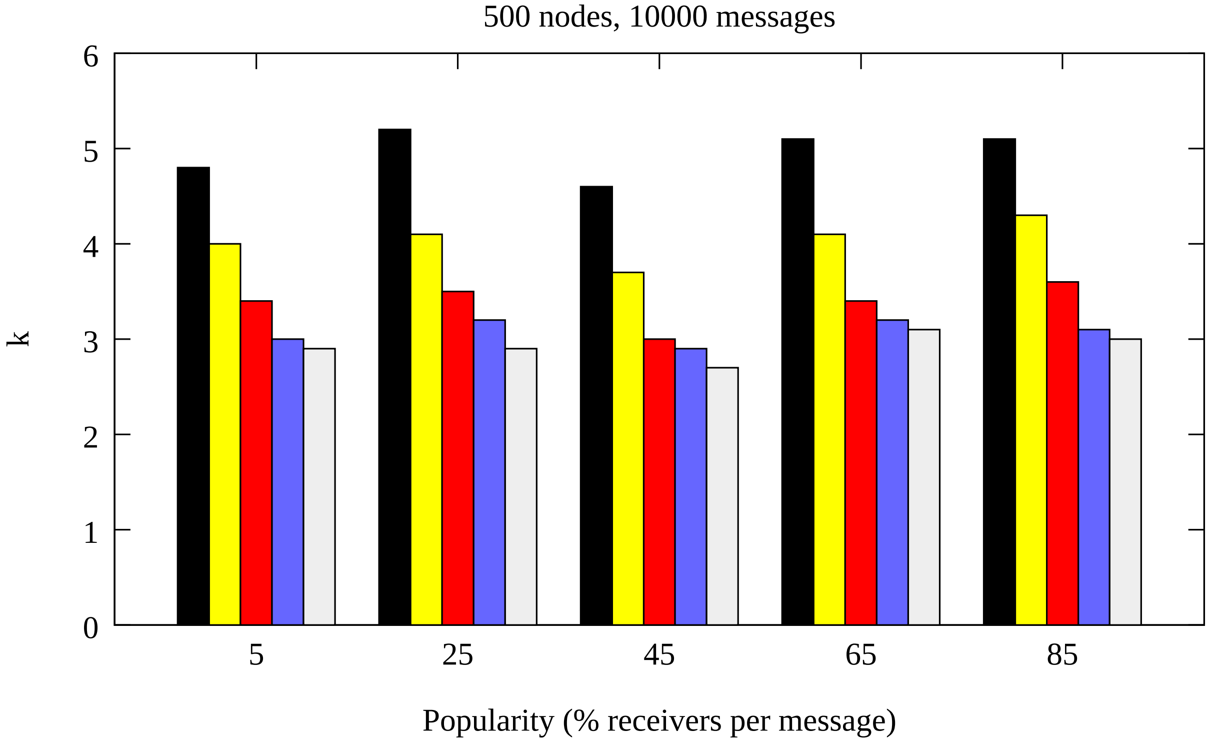

Figure 4.14: The required k to find a solution for 500 nodes and 10000 messages

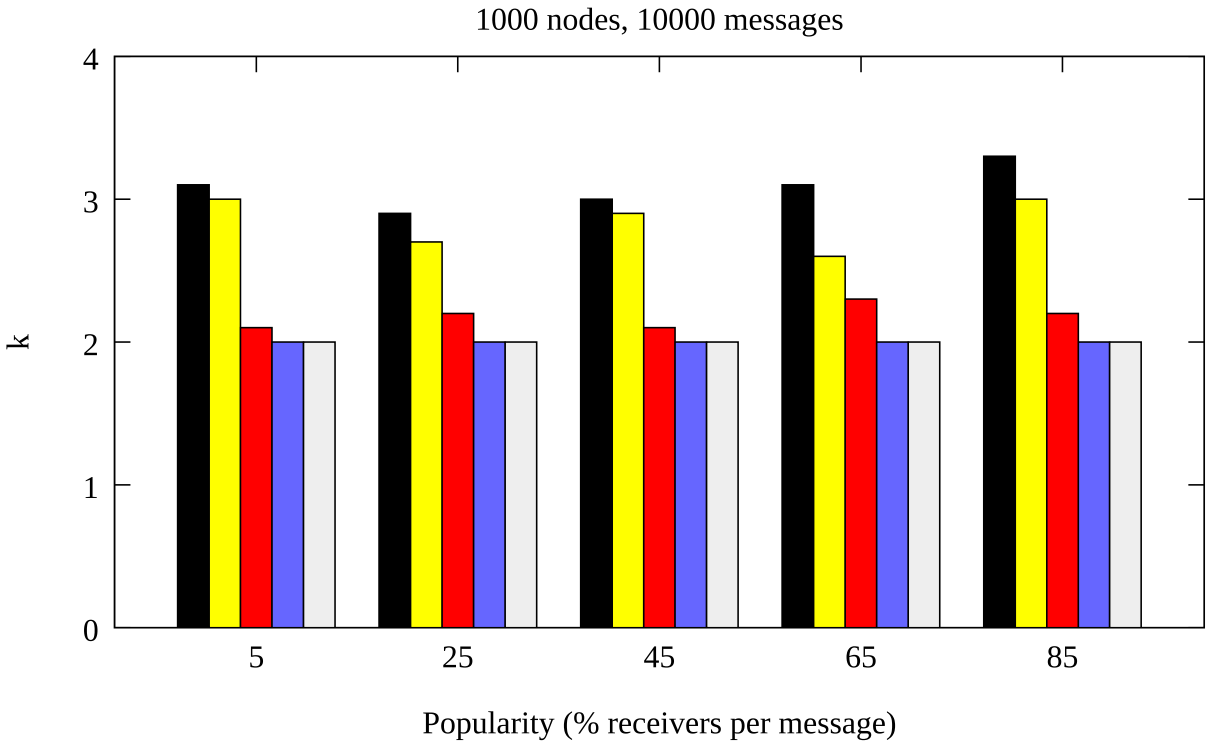

Figure 4.15: The required k to find a solution for 1000 nodes and 10000 messages

fewer k for all link capacities and popularities.

4.5.2 Solution Time

The corresponding graphs showing the time required to calculate the solutions are included. This includes the time to calculate the required paths given a topology and a workload, the time to generate a model given these paths and the time to solve the model. In most cases the time to calculate the required paths is over 99% of the total runtime so this may be considered to be the only cost.

Nevertheless, as two different models are generated, the different times taken to solve the minimum-usage model and the load-balancing model are shown separately. The graphs are organised in the same way as the graphs showing the required k in the previous section, however for each pair of link capacity and popularity value, there are two bars representing the time required for the minimum-usage model and the load-balancing model respectively. The models

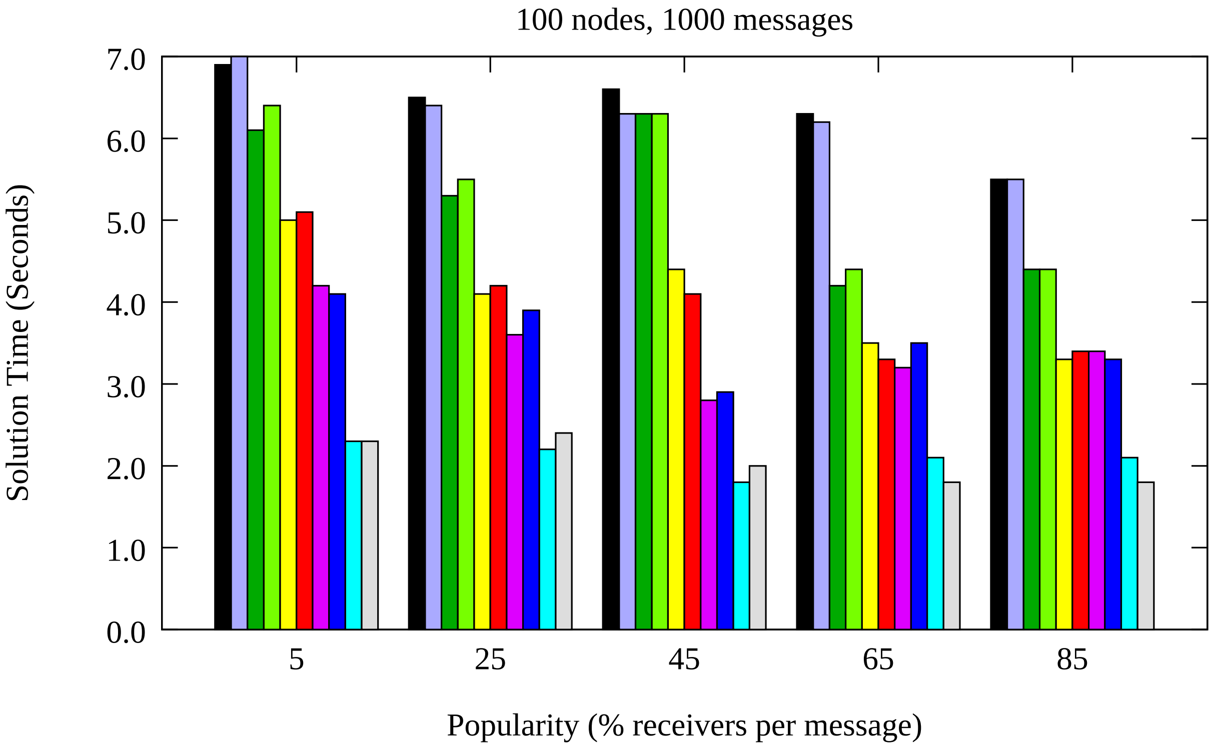

Figure 4.16: The required time to find a solution for 100 nodes and 1000 messages

just distribute messages differently over the delivery graphs and as expected there is not a big difference in the time required.

As the number of nodes increases the time required quickly increases. This is expected as it is just based on the cost of finding paths. However as the number of messages increase within one topology the time required does not change much. This is due to message splitting. Even when the number of messages increase the number of required paths to deliver them does not increase very much.

In Figure 4.16 it can be seen that for a network of 100 nodes, message popularity 65% and 20 MBit link capacity, both the minimum-usage model and the load-balancing model require just over six seconds to solve. For the same message popularity, increasing the link capacity to 60 MBits, results in a solution time of around two seconds for both models. This happens because fewer paths are required to find a solution when the capacity is greater.

Also when message popularity increases, the solution times do not change much because of

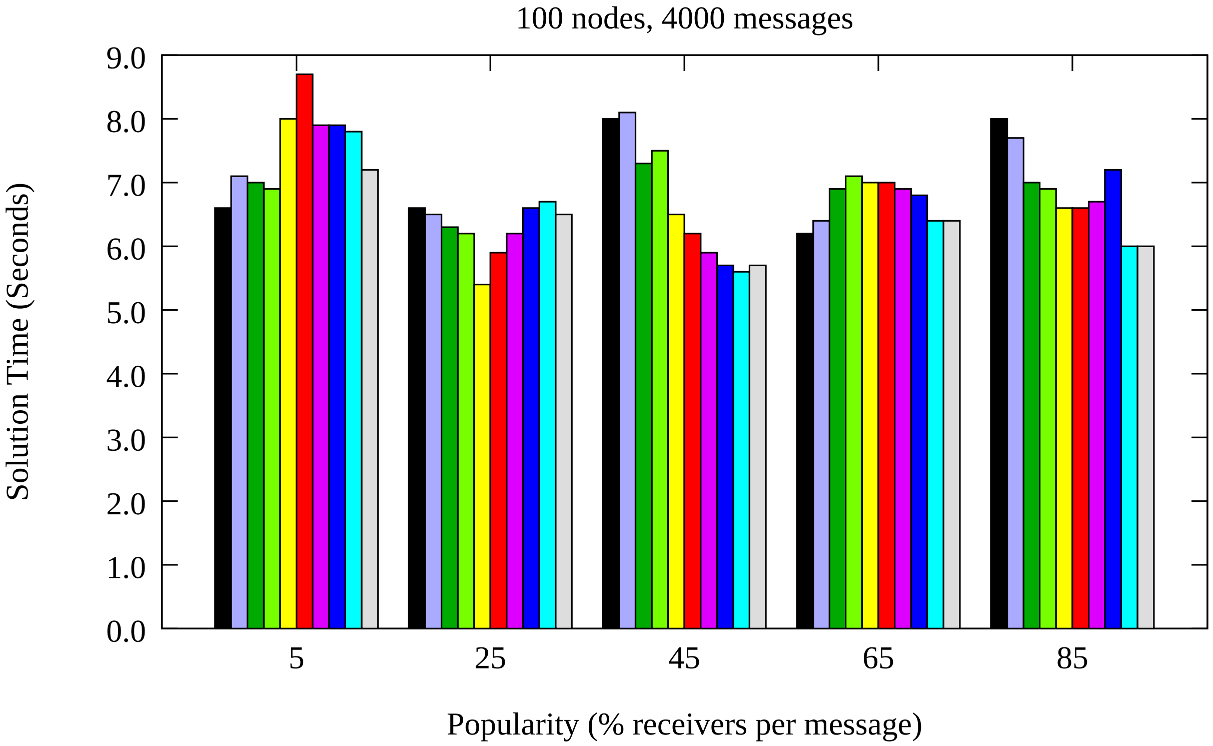

Figure 4.17: The required time to find a solution for 100 nodes and 4000 messages

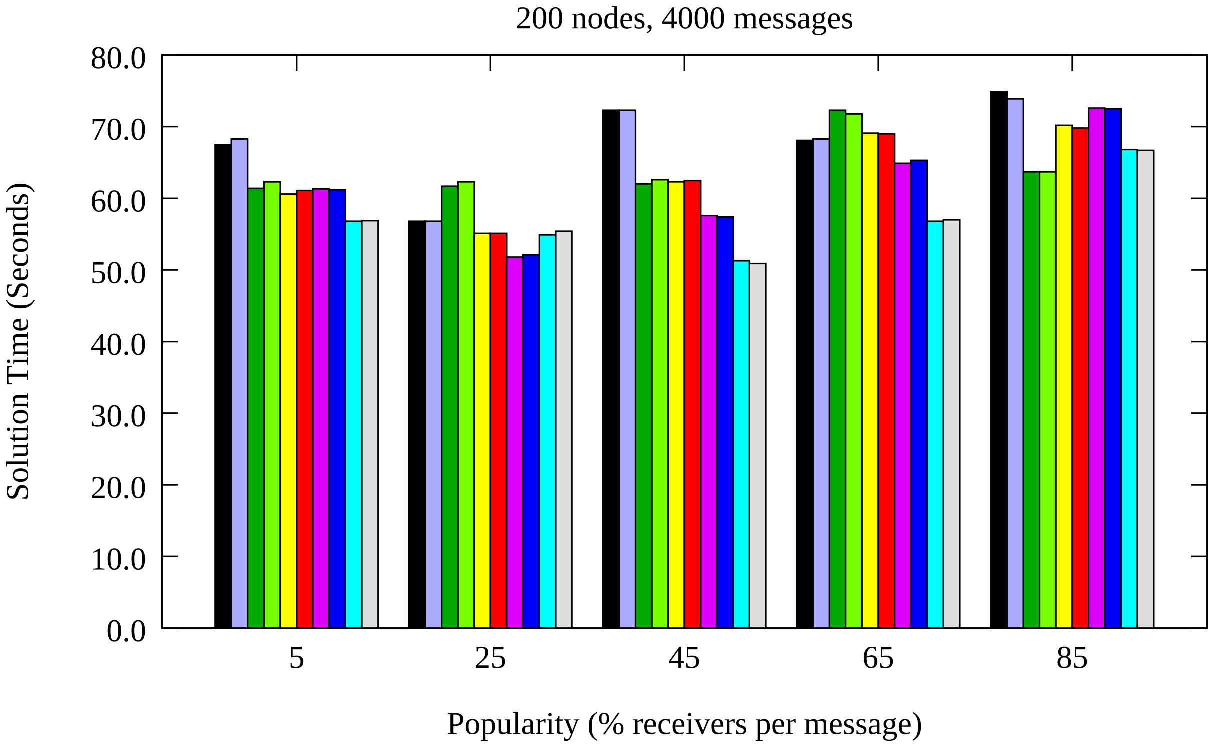

Figure 4.18: The required time to find a solution for 200 nodes and 4000 messages

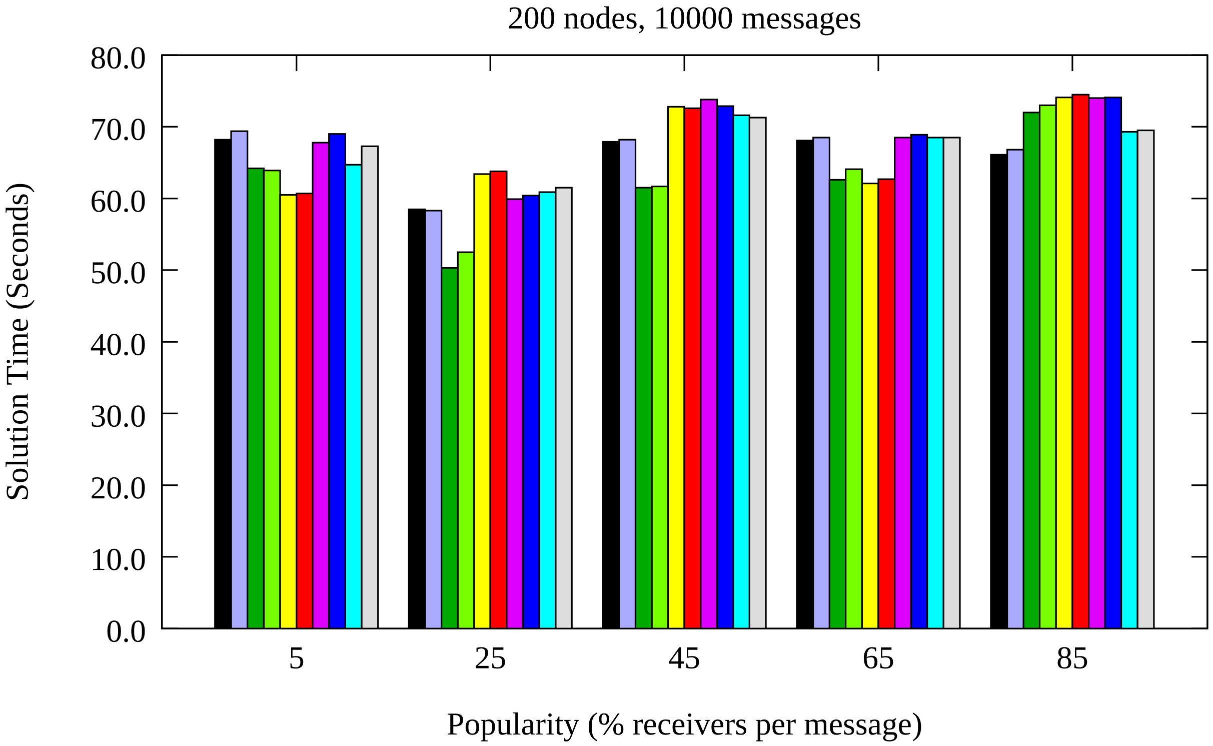

Figure 4.19: The required time to find a solution for 200 nodes and 10000 messages

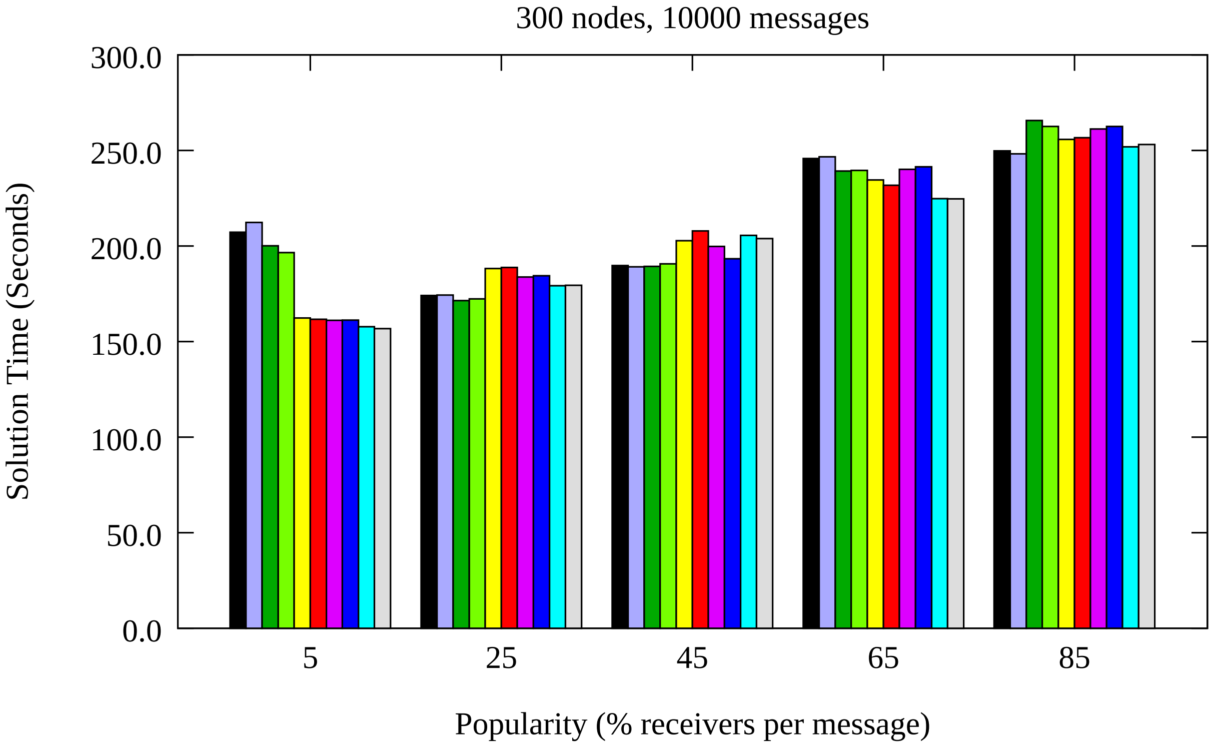

Figure 4.20: The required time to find a solution for 300 nodes and 10000 messages

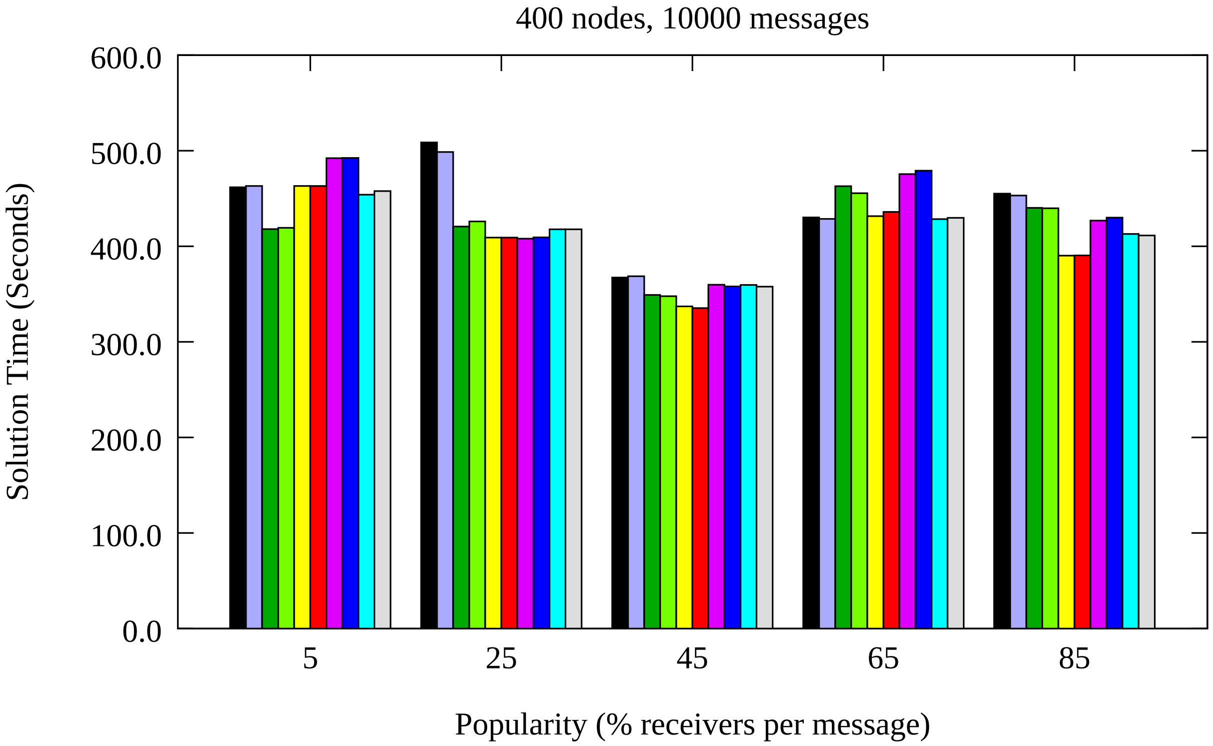

Figure 4.21: The required time to find a solution for 400 nodes and 10000 messages

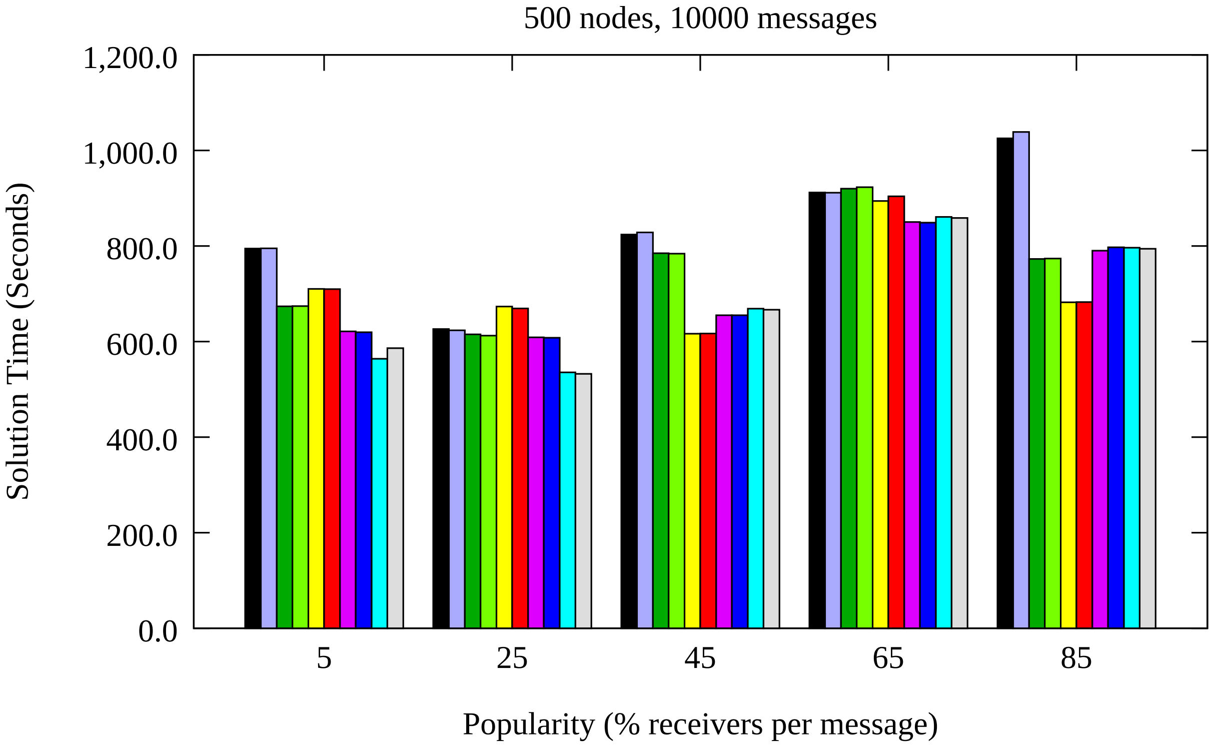

Figure 4.22: The required time to find a solution for 500 nodes and 10000 messages

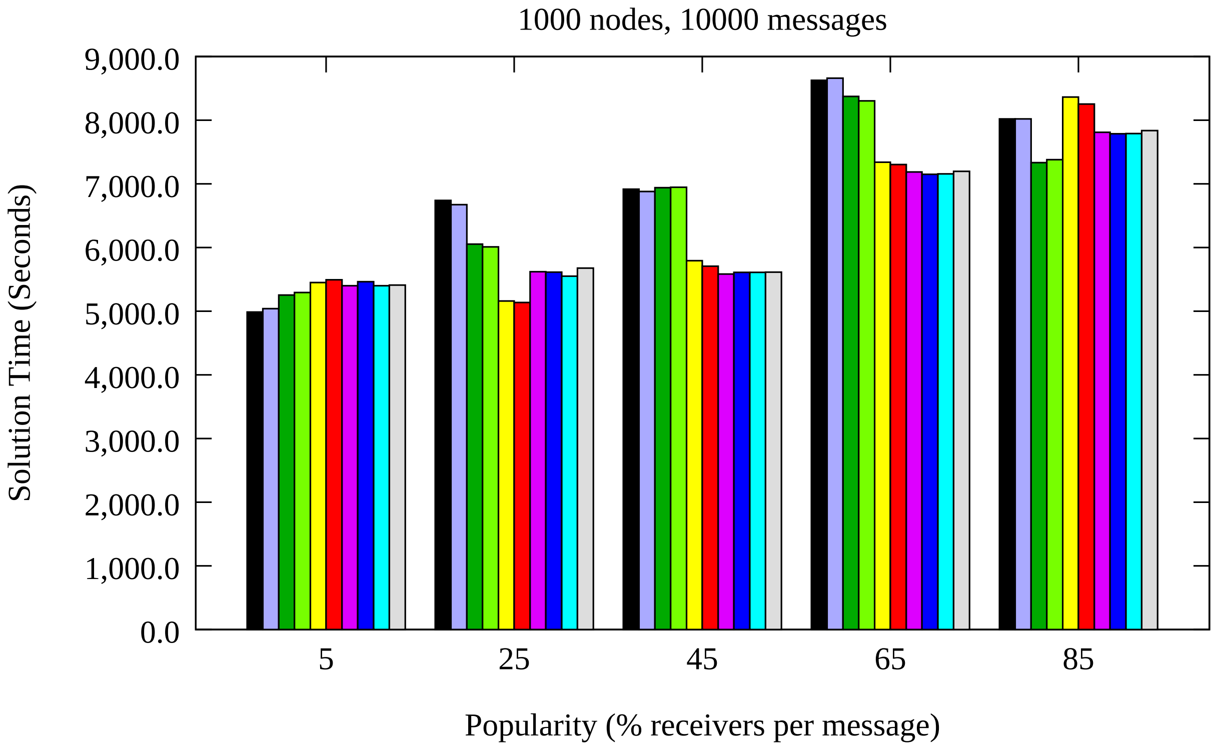

Figure 4.23: The required time to find a solution for 1000 nodes and 10000 messages

message splitting, which allows a single copy of a message to be sent to multiple receivers and copied when necessary, saving network resources.

When the number of messages in a topology of the same size increases (Figure 4.17), it can be seen that times to find solutions do not change so much, this is again due to message splitting. This can also be seen when going from 4000 to 10000 messages in the 200 nodes topology (Figures 4.18 and 4.19).

However, with the same message workload, increasing the number of nodes in the topology from 100 to 200 (Figures 4.17 and 4.18) causes the solution time to increase as finding paths takes more time. This same increase in solution time can be seen when going from 300 to 400 nodes (Figures 4.20 and 4.21) and from 500 to 1000 nodes (Figures 4.22 and 4.23).

4.6 Summary

In this chapter, the demonstration of the models representing traditional overlay routing and CAMPR were discussed. The environment and the parameter space explored were described. Graphs showing the number of undelivered messages using a traditional overlay routing approach were analysed. There was also analysis of the number of delivery graphs required to find a feasible multipath routing solution and how long finding this solution took. The results show that compared to a traditional single-path overlay routing approach CAMPR utilised network resources more efficiently, routing with fewer message drops. A simulation of CAMPR implemented in NS-3, which uses the solutions from the models to perform multicast routing with message splitting and path splitting according to the minimum usage and load-balancing goals is discussed next.

Chapter 5

Network Simulation

5.1 Introduction

As the models discussed in Section 3.6 do not take into account time, it was necessary to see if their solutions can be applied to a real routing protocol. In particular, it was necessary to test if the solutions from the CAMPR models can be used to allocate traffic and route it using path splitting and message splitting. To test if the solutions of the models can be used to build a routing protocol, a simulation using the solutions to perform routing was constructed. The simulation is a proof of concept implementation and shows that even though time is not taken into account in the models, when it does becomes a factor, the solutions from the models can still be used to construct forwarding tables and perform routing.

A discrete-event network simulator called NS-3[1] was used. This allowed for a network to be simulated without the need for a network testbed. Using NS-3, experiments under the same network conditions using the same parameters could be repeated easily. This would have been difficult to do with PlanetLab[2]. Also with NS-3, different topologies created with BRITE could be used, instead of having to use different physical network topologies. PlanetLab

[1] http://www.nsnam.org

[2] http://www.planet-lab.org

is a deployment environment rather than an experimental environment. Although this is a simulation, the implementation would not differ much from an actual prototype. Indeed this is a step towards a full implementation, which could be deployed on PlanetLab, but before this simulation needs to be done to test correctness. The simulation allowed for experiments to be run on a single machine rather than having to use a network testbed and collect results from multiple machines.

As the simulations were run on a single machine, one global forwarding table, which all nodes in the experiments refer to when forwarding messages, was used. This is the main difference between the simulation and a prototype. To make a prototype, the relevant parts of the global forwarding table would simply need to be distributed to the appropriate nodes.

The forwarding tables used in the prototype of CAMPR could potentially be bigger than those used in a traditional overlay routing approach as multiple paths between senders and receivers have to be stored. A traditional routing approach would also not need a model or model solver because only one delivery graph per message is used; there is no selection process.

However, even in the traditional approach, publishers can send their publications to multiple next hop nodes. This is not too different from the case when path splitting occurs in CAMPR. In a traditional routing approach, if the multiple next nodes are not adjacent to the publisher, a copy of the message will be sent by the publisher for each of the multiple nodes as there is no message splitting.

In this chapter, the implementation of the simulation and the results of simulating each approach are discussed.

5.2 Implementation

A modified version of the NS-3 simulator was used to simulate CAMPR. This allowed topologies generated by the BRITE topology generator to be used in NS-3. This version of NS-3 also uses a modified version of the BRITE. The output format of BRITE has been slightly altered to allow compatibility with NS-3.

In the simulations, UDP was used, however, in a prototype of CAMPR, TCP would be used for the implementation. Using UDP sockets allowed the number of undelivered messages to be calculated more easily, as they will not be retransmitted as when using TCP. However there is nothing specific to CAMPR that ties it to either UDP or TCP. In a UDP socket, when the traffic exceeds the link capacity, it is dropped. The NS-3 `FlowMonitor` class can keep track of these dropped messages, but undelivered messages need to be tracked as well and as the `FlowMonitor` class has no knowledge of subscriptions.

Before the simulation starts, the solution of the model representing the network state to be simulated has to be read in. The model solution was used to determine which messages use which paths. After this, the bottleneck-disjoint paths calculated by the model generator are read in. Besides the paths information, this is also used to determine the publishers and subscribers of the messages as the message workload passed in to the model generator is not read in.

Information such as the publisher, subscribers, delivery graph and how to distribute a message across its delivery graph is stored in the `MulticastMessage` class. A hash table of `MulticastMessage` objects acts as a global forwarding table and is used to lookup next hops each time a message is forwarded.

Finally, a BRITE configuration file which is passed to the `BriteTopologyHelper` class and from there to BRITE is read in. This is used for BRITE settings such as how many nodes to have in the topology, node placement, etc. The default settings were used here, the same as those used when generating the models. The BRITE topology returned by the `BriteTopologyHelper` class is read in, setting the link capacities in the same way as they are assigned by the model generator.

UDP sockets are created on these links and pointers to them are stored. These are then used when forwarding messages. A callback function is attached to each socket and is called whenever a message is received on that socket. NS-3 sockets are based on BSD sockets, however unlike BSD sockets, NS-3 sockets are asynchronous. Interaction between sockets and applications in NS-3 is non-blocking.

As a discrete event simulator, NS-3 allows scheduling of events at specific times in a simulation using the `Simulator::Schedule` function. Once the `Simulator::Run` function is called, the scheduled events occur at the specified times in the simulation. The maximum amount of traffic at one time is to be simulated, as this is the worst case scenario, which the models represent. Therefore, all of the messages should be published at the same time in the simulation. For all of the messages in the workload, a publishing event is scheduled at the beginning of the simulation, and this event is bound to the publishers. Once the simulation starts, all of the messages are published.

A publisher may send a message to multiple next hops and this becomes a separate message for each next hop in NS-3. It is this initial publishing of messages where path splitting could occur in the simulations of CAMPR. Once all the message transmissions have been scheduled, the simulation is started.

Apart from the publication of messages, which is scheduled beforehand, the other message transmissions are triggered when a message is received. For this, the event-based design of NS-3 was relied on. When a message is received in a socket, the callback function on the socket is called. This function extracts the message header, using the contained data to look up the next hop nodes in the forwarding table. The message header is updated to reflect the progress along the message's delivery graph and forwarding to the appropriate unique nodes is performed using those nodes' sockets.

A custom message header was created in a `MulticastHeader` class by extending NS-3's `Header` class and editing NS-3's build scripts to link in the class. In addition to the header's own functions, functions for serialization and deserialization and a few other functions from the base `Header` class had to be implemented. The message number is stored, as well as how far

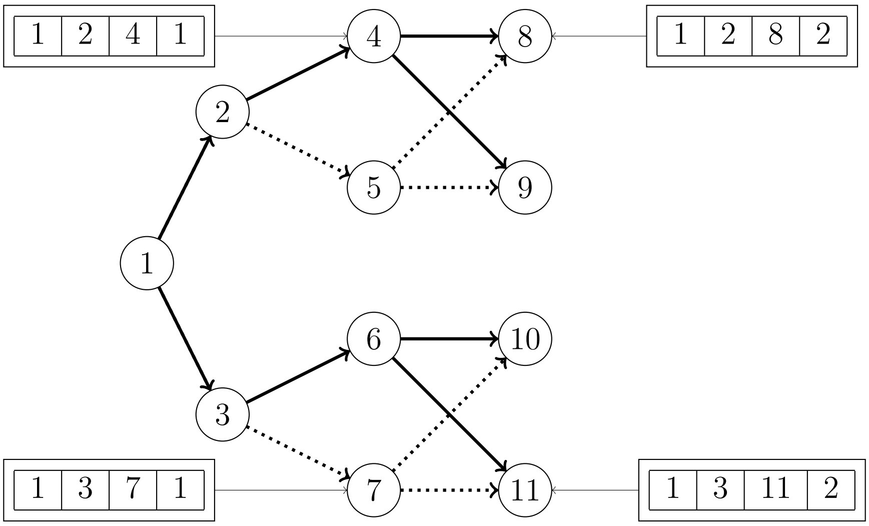

Figure 5.1: Two delivery graphs with the message headers shown at nodes 4, 7, 8 and 11

Forwarding Table		
Delivery Graph	Next Hop	Path
1	2	2 4 8
		2 4 9
	3	3 6 10
		3 6 11
2	2	2 5 8
		2 5 9
	3	3 7 10
		3 7 11

Figure 5.2: Forwarding table for the situation shown in Figure 5.1

along it is in the current path from the sender to the receiver.

The current progress along the path does not need to be sent in the message header, but this allows for a faster lookup in the forwarding table. When a message is received, the header is extracted and the information is used to look up the next hops along the path. The necessary fields are then updated and the message is forwarded to the required nodes.

Figure 5.1 shows a small topology which will be used to illustrate the forwarding process. Assume that node 1 will send a message (call this message 1) to nodes 8, 9, 10 and 11 along two delivery graphs. Assume that this message will be divided evenly among the delivery graphs. In the model solution, this situation would be represented by the variables `x_1_1 = 0.5` and `x_1_2 = 0.5`. Meaning that half of message 1 will be sent down delivery graph 1 and the other half of message 1 will be sent down delivery graph 2. The first delivery graph is shown in a thicker line and the second delivery graph is shown in a dotted line. The two delivery graphs overlap at links $1 \rightarrow 2$ and $1 \rightarrow 3$.

The associated forwarding table is show in Figure 5.2. The paths are calculated from the sender to each receiver. This information is stored with message 1. The forwarding table is divided into delivery graphs and then further into the next hop node from the publisher. This is because in NS-3 when publishing messages, a message needs to be created for each next hop of a publisher. Another reason for this is to reduce the lookup time in the forwarding table when forwarding messages.

The message header contains the message number, the next hop node from the publisher, the node to which the message will be forwarded and the index of this node in the forwarding table. The delivery graph number is stored with the message, but is not included in the message header. At each hop this header is extracted and the forwarding table is used to look up the next hops.

Figure 5.1 shows the message header at several nodes. At node 4 the message header contains the values $\{1, 2, 4, 1\}$. Meaning that this is message 1, the next hop node from the publisher is node 2, the current node is node 4, and its index in the paths is 1. Note that the paths also contain the next hop node from the publisher and the message header starts off at index 0. Node 4 will extract this header and look up delivery graph 1, next hop 2 and index 1 in the paths stored in the forwarding table to find out that it should forward this message to nodes 8 and 9.

There are multiple unique next hops, meaning that message splitting will occur at this node. The path index will be incremented to 2 and two messages will be created, one with the header's current node set to 8 and one with the header's current node set to 9. The first two fields of the message header will not change.

The header in the message received at node 8 can be seen in Figure 5.1. Note that the header of the message sent along delivery graph 2 will be the same at node 8. Once node 8 extracts the header of the message it will check the forwarding table and realise that it is the last node in the path, therefore a subscriber. The message headers are also shown for delivery graph 2 at nodes 7 and 11.

Note that there may be paths with the same next hop, but only a single copy of messages is forwaded to unique nodes. Message are copied only when the next hop nodes are different. This is the implementation of message splitting. Readers may be wondering about the case when two or more paths have a different next hop node, but they intersect later on. In this case, the part of the path before the intersection could be eliminated for all but one of the paths, and message splitting could be used at the intersection.

However, due to the shortest paths algorithm used, this case does not occur. The shortest paths are determined by starting at the leaves of the shortest-paths tree and the candidate paths converge towards this tree as the root is approached. Any overlap among the selected paths will start at the root and they do not intersect later on.

After the simulation ends, clean up is performed, statistics are collected from the `FlowMonitor` class and the number of undelivered messages is counted. This needs to take into account the possibility of path splitting. When a message is distributed across multiple delivery graphs, if a subscriber is missing any of the pieces of a message, this message will be considered to be undelivered. Now the results are discussed.

5.3 Simulation Results

In this section, the results of simulating CAMPR are discussed. The two goals of CAMPR are minimum usage and load-balancing. Table 5.1 shows the parameters used for both the minimum usage and the load-balancing simulations. Note that the 5000 message workload was only used with the 100 node topology, while the other workloads were used for all the topologies.

Parameter	Range
Popularity (% receivers)	5, 25, 45, 65, 85
Workload size (number of messages)	5000, 10000
Link capacity (Mbits)	20, 30, 40, 50, 60
Topology size (number of nodes)	100, 200, 300, 400, 500, 1000

Table 5.1: Parameters used and the ranges of their values

These are the same parameters as those used in Section 4.4. Also, the same seeds were used to determine which nodes are the publishers when running these simulations.

For the minimum usage goal there were no undelivered messages. There were no solutions of the minimum usage models which used multiple delivery graphs per message (only a single path was used in all cases). This is expected as the minimum usage goal aims to minimise network usage, i.e., the amount of nodes used to route a publication. For the load-balancing goal there were also no undelivered messages. However, routing according to the solutions of the load-balancing models do use multiple delivery graphs per message as the goal is to spread the load as evenly as possible.

5.4 Summary

In this chapter, it was shown that the solutions from the CAMPR models can be used to perform routing with path splitting and message splitting. A simulation of CAMPR was created, but this can be used to create a prototype of the routing protocol as well. For this simulation, UDP sockets were used between nodes. This allowed for easy checking of undelivered messages, as dropped packets were not retransmitted, but for a prototype TCP sockets would be used.

Chapter 6

Conclusion

A method for avoiding congestion faced in overlay networks was presented. Selecting a single latency-shortest path between a sender and receiver was avoided because this greedy approach to path selection can cause a build up of packets at nodes connected to low-latency links, as many packets will be routed through them. As these nodes may not have enough processing power to process so many packets, they may end up dropping them, causing retransmissions and delaying delivery.

One way of dealing with this problem is through congestion control. These schemes measure available bandwidth or check for dropped packets to detect congestion. Measuring available bandwidth cannot be done accurately and cheaply and dropping packets was to be avoided. In the routing approach, congestion was avoided by routing messages through nodes which have available resources.

Senders were allowed to split messages across several paths going to a sender. This multipath routing scheme has been shown to increase throughput [BO07, WWK^{+}07]. Using multiple paths meant that there were more network resources available for sending a flow of messages, reducing congestion. This was done to avoid rate limiting and having to retransmit dropped packets.

Multipath routing protocols also split message flows across multiple paths, but a somewhat unique approach was used for path selection. Paths were ranked by latency, but from these paths, those which are bottleneck-disjoint with respect to capacity were selected. This increased the capacity available to a message flow while still maintaining speed along the paths.

The multipath routing protocols mostly deal with sender-receiver pairs and are not multicast like CAMPR. There are centralised and distributed approaches. A centralised approach was used, assuming global knowledge of the network. Global solutions are more accurate, but a distributed solution is also attractive for cases when it is infeasible to acquire global knowledge of the network. A distributed solution is one possible area of future work.

Another related field is traffic engineering which is a technique used by ISPs to make the most of network resources. In this field there are also distributed and centralised approaches, but the main goal is to load-balance traffic and minimise congestion as in this work. As multipath routing approaches, traffic engineering schemes are often cast into optimisation problems, which is the approach taken in this work.

The field of bandwidth reservation is also related because it attempts to use network resources efficiently by allocating requests for traffic such that certain QoS is provided to applications. This is a similar goal to that of this work. Approaches in this field use multipath routing and can also be used together with traffic engineering.

A unique characteristic of this approach, not seen in combination with the work done in multipath routing and traffic engineering, is message splitting. This saves network resources by having nodes duplicate messages they have received as close as possible to the receivers. The problem considered was the optimising of multipath routing taking into account all of the senders in the network, sending messages to multiple receivers and taking advantage of message splitting. This global solution required global knowledge of the network, but many approaches in multipath routing and traffic engineering make similar assumptions so this is a reasonable assumption.

6.1 Summary of Thesis Achievements

A model of a multicast messaging middleware which allows message splitting and path splitting was presented. The model generator created takes a network topology and a message workload and generates a model of this, representing the state of the network including the available resources of the network. The generator computes bottleneck-disjoint paths from the senders to the receivers and then uses these paths to build the delivery graphs for each message flow. To the best of my knowledge this is the first such model that considers message splitting and path splitting.

When solved, the model finds a routing approach which results in a message flow distribution that avoids dropping messages with the given resources in the network. This solution can be used to construct forwarding tables and route and split messages accordingly. The routing scheme presented along with the bottleneck-disjoint path selection technique can be applied to any overlay network. In fact, the model can be used to model any multicast network with path splitting and message splitting.

6.2 Applications

The routing approach can be applied to any type of network. Only the sources and destinations of the messages and the topology need to be input to the model generator and a corresponding model will be generated, allowing for creating of the forwarding tables.

As global knowledge of the network has been assumed, this approach will only work for small networks or for networks in which similar assumptions can be made. One example is a data centre, where a centralised approach is appropriate. In small networks this approach can be used in semi real-time to make routing decisions. As the time to calculate paths increases with the size of the network, in larger networks the routing decisions have to be made offline.

Another possibility is to make an estimate of the global state of the network and present this to the model generator. This will not result in as good a solution as if complete knowledge was available, but it might still be useful. Some multipath routing approaches looked at use a hybrid approach. The available paths among which traffic can be split are determined using global knowledge, but the distribution decisions are made using local knowledge.

6.3 Limitations

The main limitation of this work is the fact that global knowledge of the network is required. This assumption was made in order to be able to generate an accurate model of the network including all of the messages. This centralised approach leads to the best results possible for the given topology and message load, but assuming global knowledge is often unrealistic in real networks. This was just the first step in this approach to avoiding congestion.

The fact that the model is computed for the entire network leads to another limitation. Generating the model requires computation of the delivery graphs of all the messages. Only after all of the paths have been calculated, can the forwarding tables of nodes in the network begin to be constructed. It would be better to be able to compute the delivery graphs of messages independently of each other. This would allow messages to be sent in real-time as they appear in the network instead of having to wait for the delivery graphs of the all the messages to be calculated as in the current offline approach.

Based on the approach described so far, heuristics that approximate the current solution can be created. A possible way is to keep track of the popularity of specific content in messages flowing through nodes as in a tag cloud and use this to predict capacity requirements. A node which publishes popular content will be expected to generate a lot of outgoing traffic while a node subscribed to popular content to receive a lot of traffic. A node which sees many messages containing some particular content flowing through it may detect this as popular content and signal to the network that a path split should be done with that node being the bottleneck node which to avoid in the new path. By anticipating traffic patterns in this way, congestion may be alleviated without having full knowledge of the network.

The scalability of this approach is limited by the path-finding algorithm which accounts for about 99% of the time taken to find a routing solution given a network and a message workload. As this is a polynomial algorithm, the current approach becomes quite slow when there are 1000 nodes, as can be seen in Section 4.5.2. Experiments were run using workloads with up to 10000 messages, but this approach could scale to more messages. The limiting factor is the number of nodes in the network rather than the number of messages. As can be seen from the graphs showing solution times, the number of messages does not have much effect on the time taken to find a solution. Note that, for this approach the path finding was not performed in parallel. If the path finding were to be parallelised and distributed, scalability could be increased.

6.4 Future Work

There are several paths that can be explored in the future. The most immediate one is to develop a distributed version of the models, allowing a solution to be generated without having complete knowledge of the network. This could lead to an on-line solution with real-time updates to forwarding tables, dynamically adapting to user subscriptions and network conditions. If global knowledge of the network is no longer assumed, the solution will become less accurate, but some measures can be taken to improve the efficiency of the routing.

One technique which has been shown to improve network throughput is network coding. This is a technique for data delivery in networks in which intermediate nodes along the path from source to destination are allowed to perform operations on the packets in addition to just relaying them. Several packets can be encoded together and when delivered, the receiver can use the messages it has received so far to decode the message. This works well in a publish-subscribe context as messages with the same or similar content can be encoded and users with matching subscriptions will have previous messages containing this content, allowing them to decode the new messages.

The encoding of messages can be as simple as XOR-ing them together as in [KRH$^+$08] or more complicated such as computing a linear combination of chunks of the messages and sending the coefficients as well as in [BA09]. Previously encoded messages can also be encoded. In any case, the encoding and decoding processes need to be cheap and scale with the size of messages.

It might be the case that a node does not have enough information to decode a particular message. In this case it needs to seek the missing messages, starting by asking nodes near to it. Users with similar interests tend to be close together in a network, so this is also suitable for a publish-subscribe-based network.

A possible way to prevent this is to keep track of which messages have been sent to neighbours and only encoded messages in such a way that neighbours can decoded them. This can also be used to make forwarding decisions, picking neighbours which are known to be able to decode the message. However, these kind of decisions should not delay the sending of messages. If there is no suitable encoding to be made, the plain unencoded message should be sent.

As in [KRH$^+$08], small and large messages can be kept track of, and messages with similar sizes can be encoded to save the most bandwidth. This paper also showed that there is a much higher throughput gain when using UDP than when using TCP. This is because the congestion control of TCP matches the input rate of a node to its output rate. Therefore in TCP senders back off to prevent packet dropping. With UDP this does not happen. Another interesting point is that without network coding, network fairness and efficiency are conflicting goals, but with network coding increasing fairness increases the throughput achieved in the network.

Another area that can be explored is adding features to the model itself. This can include a goal which is a combination of the two current goals, minimum usage and load-balancing. These goals can be combined with a weight assigned to each, allowing the user to tweak the optimisation to their specific needs. Another possibility is adding priorities to the traffic, by assigning a weight to each flow. This may be done manually by the user or perhaps the user can specify, for example, that smaller flow are to be favoured over larger ones and the model generator can assign suitable weights to the flows.

6.5 Summary

Motivations for a congestion-avoiding messaging middleware were given and work done in the area of messaging middleware was reviewed. Additionally, work done in the related field of content-based networking was also reviewed.

Techniques such as backpressure, used in congestion control schemes in overlay networks, were looked at. These techniques use rate limiting to slow down the transmission of packets once congestion has been detected. As congestion was to be avoided altogether, work done in the field of multipath routing, a technique in which a message flow is split across multiple paths from sender to receiver, thus increasing the capacity to the message flow, was looked at.

The principles of multipath routing were applied to this work, message flows were split across multiple delivery graphs. Furthermore, a bottleneck-disjoint technique was used to select paths across which to distribute traffic.

Work done in traffic engineering was also discussed because it has a similar goal to this work, avoiding congestion and load-balancing network traffic. The routing decision was cast as an optimisation problem, an approach frequently used in this field.

The field of bandwidth reservation - a technique for managing network resources such that a certain QoS is provided to user applications, was also looked at. This can be used in conjunction with traffic engineering and multipath routing. Heuristic approaches are frequently used to make the bandwidth reservations.

In this work, multiple senders (publishers) send messages to multiple receivers (subscribers) in the context of a store-and-forward network. There may be many or no subscribers for each message. The aim was to optimise global routing, avoiding congestion and overloading the network. Some assumptions needed to be made to make this a tractable problem: complete knowledge of the topology and the messages in the network. These assumptions are common in publish-subscribe networks.

The state of the network, publishers, subscribers and messages were represented as a mathematical model, using an optimisation solver to solve it. The solution of the model describes how to split message flows across their delivery graphs. This information can be used to construct the forwarding tables in a routing protocol. Generating the model involved calculating possibly multiple paths between all the senders and receivers, which accounted for almost all of the time required for model generation.

A small topology was used as an example to informally describe how the approach works in scenarios of increasing complexity and then formally defined the concepts. The overall goals of maximum flow and minimum latency were pursued. Maximum flow has the subgoals of minimising network usage, using as few links as possible and load-balancing, evenly distributing messages across as many links as possible. The goal of minimum latency was treated as a secondary goal. Delivery graphs with enough capacity for a particular message flow were used over a faster delivery graph.

Different path selection techniques attempted were discussed as well as why eventually the capacity based bottleneck-disjoint approach was chosen. The mathematical models generated using the selected paths and the state of the network were defined. This included the minimum network usage model and the load-balancing model along with the optimal versions of these. Solving the optimal models is intractable, but they were defined them so that the reader understands what was aimed for. A model which counts the number of undelivered messages when using a traditional overlay routing approach was also defined.

The evaluation framework used was described, including the evaluation process, the parameter space explored and the environment used. Results from the minimum usage and load-balancing models were presented, showing how many paths it took to find a feasible routing solution given a message workload and a topology. Results showing the time taken to find these solutions were also included. In addition, results from the traditional overlay routing model showing the number of undelivered messages for various workloads and topologies were presented. Only a subset of the graphs was included as the parameter space explored is quite large.

The implementation of a simulation of CAMPR was explained, showing that multipath routing can be performed according to the solutions of the models using path splitting and message splitting. The implementation of message splitting and path splitting were described, as well as how the forwarding table is constructed. The simulation was implemented to verify that the models can be in a real routing protocol, as the models do not take into account time. With minimal changes it can be used to create a prototype of CAMPR.

An overview of the work was provided, discussing the contributions and possible areas in which it can be applied. The limitations of the work were also discussed, mainly the fact that it requires global network knowledge and is an offline approach. Possible avenues for future work such as distributing the model in order to find quicker solutions and introducing network coding to further increase throughput were also discussed.

Bibliography

[ABIS06] T. Anjali, C. Bruni, D. Iacoviello, and C. Scoglio. Dynamic bandwidth reservation for label switched paths: An on-line predictive approach. *Comput. Commun.*, 29(16):3265–3276, October 2006.

[ABKM01] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *SOSP*, pages 131–145, 2001.

[AMG05] N. AbuAli, H.T. Mouftah, and S. Gazor. Multi-path traffic engineering distributed vpls routing algorithm. In *Systems Communications, 2005. Proceedings*, pages 275–280, August 2005.

[ASB03] David G. Andersen, Alex C. Snoeren, and Hari Balakrishnan. Best-path vs. multi-path overlay routing. In *Internet Measurement Conference*, pages 91–100. ACM, 2003.

[ASB10] Stefan Appel, Kai Sachs, and Alejandro Buchmann. Towards benchmarking of amqp. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, DEBS '10, pages 99–100, New York, NY, USA, 2010. ACM.

[ASD09] Habtamu Abie, Reijo M Savola, and Ilesh Dattani. Robust, secure, self-adaptive and resilient messaging middleware for business critical systems. In *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD '09. Computation World:*, pages 153–160, November 2009.

[BA09] Roberto Beraldi and Hussein M. Alnuweiri. A network-coding based event diffusion protocol for wireless mesh networks. In Athanasios V. Vasilakos, Roberto Beraldi, Roy Friedman, and Marco Mamei, editors, *Autonomics*, volume 23 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 17–31. Springer, 2009.

[BBQV07] R. Baldoni, R. Beraldi, L. Querzoni, and A. Virgillito. Efficient publish/subscribe through a self-organizing broker overlay and its application to Siena. *The Computer Journal*, 50(4), 2007.

[BFS09] O. Boyaci, A.G. Forte, and Henning Schulzrinne. Performance of video-chat applications under congestion. In *Multimedia, 2009. ISM '09. 11th IEEE International Symposium on*, pages 213–218, Dec 2009.

[Bit06] Sven Bittner. Supporting arbitrary boolean subscriptions in distributed publish/ subscribe systems. In *MDS '06: Proceedings of the 3rd international Middleware doctoral symposium*, New York, NY, USA, 2006. ACM.

[BMVV05] Roberto Baldoni, Carlo Marchetti, Antonino Virgillito, and Roman Vitenberg. Content-based publish-subscribe over structured overlay networks. In *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, pages 437–446, Washington, DC, USA, 2005. IEEE Computer Society.

[BO07] R. Banner and A. Orda. Multipath routing algorithms for congestion minimization. *Networking, IEEE/ACM Transactions on*, 15(2):413–424, 2007.

[Bro11] Paul C. Brown. *TIBCO Architecture Fundamentals.* Addison-Wesley, Boston, MA, USA, 2011.

[CCK$^+$08] Kideok Cho, Jaeyoung Choi, Dong-Il D. Ko, Taekyoung Kwon, and Yanghee Choi. Content-oriented networking as a future internet infrastructure: Concepts, strengths, and application scenarios. In *3rd International Conference on Future Internet Technologies (CFI08)*, Seoul, Korea, June 2008.

[CCP07] Stefano Castelli, Paolo Costa, and Gian Pietro Picco. Modeling the communication costs of content-based routing: the case of subscription forwarding. In *DEBS '07: Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems*, pages 38–49, New York, NY, USA, 2007. ACM.

[CCP08] S. Castelli, P. Costa, and G.P. Picco. Hypercbr: Large-scale content-based routing in a multidimensional space. *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1714–1722, April 2008.

[CER12] Sangman Cho, T. Elhourani, and S. Ramasubramanian. Independent directed acyclic graphs for resilient multipath routing. *Networking, IEEE/ACM Transactions on*, 20(1):153–162, February 2012.

[CGMP13] Giovanna Carofiglio, Massimo Gallo, Luca Muscariello, and Michele Papalini. Multipath congestion control in content-centric networks. *IEEE INFOCOM Workshop on Emerging Design Choices in Name-Oriented Networking*, April 2013.

[CJT01] Luis-Felipe Cabrera, Michael B. Jones, and Marvin Theimer. Herald: Achieving a global event notification service. In *HotOS*, pages 87–92. IEEE Computer Society, 2001.

[CRW06] Antonio Carzaniga, Aubrey J. Rembert, and Alexander L. Wolf. Understanding content-based routing schemes. Technical Report 2006/05, Faculty of Informatics, University of Lugano, September 2006.

[CRW09] Antonio Carzaniga, Matthew J. Rutherford, and Alexander L. Wolf. A memory-efficient per-source routing scheme for content-based networks. Unpublished paper, 2009.

[CW01] Antonio Carzaniga and Alexander L. Wolf. Content-based networking: A new communication infrastructure. In *NSF Workshop on an Infrastructure for Mobile* *and Wireless Systems*, number 2538 in Lecture Notes in Computer Science, pages 59–68, Scottsdale, Arizona, October 2001. Springer-Verlag.

[CW03] Antonio Carzaniga and Alexander L. Wolf. Forwarding in a content-based network. In *Proceedings of ACM SIGCOMM 2003*, pages 163–174, Karlsruhe, Germany, August 2003.

[DGD05] Zhenhai Duan, Kartik Gopalan, and Yingfei Dong. Push vs. pull: implications of protocol design on controlling unwanted traffic. In *SRUTI'05: Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop*, Berkeley, CA, USA, 2005. USENIX Association.

[dQVMdS00] Ernesto de Queirós Vieira Martins and José Luis Esteves dos Santos. A new shortest paths ranking algorithm. *Investigao Operacional*, 20:47–62, 2000.

[EEM04] Viktor S. Wold Eide, Frank Eliassen, and Jorgen Andreas Michaelsen. Exploiting content-based networking for video streaming. In *MULTIMEDIA '04: Proceedings of the 12th Annual ACM International Conference on Multimedia*, pages 164–165, New York, NY, USA, 2004. ACM.

[EJLW01] A. Elwalid, C. Jin, S. Low, and I. Widjaja. Mate: Mpls adaptive traffic engineering. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1300–1309, 2001.

[ER04] T. Erlebach and M. Ruegg. Optimal bandwidth reservation in hose-model vpns with multi-path routing. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2275–2282, 2004.

[ETR$^+$13] Roberto R. Expósito, Guillermo L. Taboada, Sabela Ramos, Juan Touriño, and Ramon Doallo. Evaluation of messaging middleware for high-performance cloud computing. *Personal and Ubiquitous Computing*, 17(8):1709–1719, 2013.

[GE09] Luis Garcés-Erice. Building an enterprise service bus for real-time soa: A messaging middleware stack. In Sheikh Iqbal Ahamed, Elisa Bertino, Carl K. Chang, Vladimir Getov, Lin Liu, Hua Ming, and Rajesh Subramanyan, editors, *COMPSAC (2)*, pages 79–84. IEEE Computer Society, 2009.

[HBCR07] J. He, M. Bresler, M. Chiang, and J. Rexford. Towards robust multi-layer traffic engineering: Optimization of congestion control and routing. *Selected Areas in Communications, IEEE Journal on*, 25(5):868–880, June 2007.

[HSH$^+$06] Huaizhong Han, Srinivas Shakkottai, C. V. Hollot, R. Srikant, and Donald F. Towsley. Multi-path tcp: a joint congestion control and routing scheme to exploit path diversity in the internet. *IEEE/ACM Trans. Netw.*, 16(6):1260–1271, 2006.

[HWJ05] Junghee Han, David Watson, and Farnam Jahanian. Topology aware overlay networks. In *INFOCOM*, pages 2554–2565. IEEE, 2005.

[HZZ$^+$09] Jianqiang Hu, Zhiyong Zeng, Geke Zhao, Chenfeng Long, and FengE Luo. s-amm: A service-oriented asynchronous messaging middleware. In *Electronic Commerce and Security, 2009. ISECS '09. Second International Symposium on*, volume 2, pages 312–316, May 2009.

[IN02] Rauf Izmailov and Dragos Niculescu. Flow splitting approach for path provisioning and path protection problems. In *High Performance Switching and Routing, 2002. Merging Optical and IP Technologies. Workshop on*, pages 93–98, 2002.

[IUY06] T. Ishida, K. Ueda, and T. Yakoh. Fairness and utilization in multipath network flow optimization. In *Industrial Informatics, 2006 IEEE International Conference on*, pages 1096–1101, August 2006.

[JBB12] Yue Jia, E. Bodanese, and J. Bigham. Checking the robustness of a message oriented middleware based system. In *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2012 4th International Congress on*, pages 280–285, Oct 2012.

[JF08] Zbigniew Jerzak and Christof Fetzer. Bloom filter based routing for content-based publish/subscribe. In *DEBS '08: Proceedings of the second international Conference on Distributed Event-Based Systems*, pages 71–81, New York, NY, USA, 2008. ACM.

[JMSGLA07] V. Jacobson, M. Mosko, D. Smetters, and J. Garcia-Luna-Aceves. Content-centric networking. White paper, Palo Alto Research Center, January 2007.

[JSB$^+$09] Van Jacobson, Diana K. Smetters, Nicholas H. Briggs, Michael F. Plass, Paul Stewart, James D. Thornton, and Rebecca L. Braynard. Voccn: voice-over content-centric networks. In *ReArch '09: Proceedings of the 2009 Workshop on Re-architecting the internet*, pages 1–6, New York, NY, USA, 2009. ACM.

[JSK$^+$11] Vijay Jain, Rajeev Srivastava, Ranjan Kumar, Rahul Upadhyay, and Kapil Kant Kamal. Breaking barrier to technology: e-governance messaging middleware. In Elsa Estevez and Marijn Janssen, editors, *ICEGOV*, pages 273–276. ACM, 2011.

[JST$^+$09] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *CoNEXT '09: Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, pages 1–12, New York, NY, USA, 2009. ACM.

[KGL00] M. Karol, S.J. Golestani, and D. Lee. Prevention of deadlocks and livelocks in lossless, backpressured packet networks. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1333–1342, 26-30 2000.

[KKDC05] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. Walking the tightrope: responsive yet stable traffic engineering. *SIGCOMM Comput. Commun. Rev.*, 35(4):253–264, August 2005.

[KKY+10] Minkyong Kim, Kyriakos Karenos, Fan Ye, Johnathan Reason, Hui Lei, and Konstantin Shagin. Efficacy of techniques for responsiveness in a wide-area publish/subscribe system. In *Proceedings of the 11th International Middleware Conference Industrial Track*, Middleware Industrial Track '10, pages 40–45, New York, NY, USA, 2010. ACM.

[Kra09] Joshua Kramer. Advanced message queuing protocol (amqp). *Linux J.*, 2009(187), November 2009.

[KRH+08] S. Katti, H. Rahul, Wenjun Hu, D. Katabi, M. Medard, and J. Crowcroft. Xors in the air: Practical wireless network coding. *Networking, IEEE/ACM Transactions on*, 16(3):497–510, 2008.

[LBHO05] Chansook Lim, S. Bohacek, J.P. Hespanha, and K. Obraczka. Hierarchical max-flow routing. In *Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE*, volume 1, December 2005.

[LBS+08] Sung-Ju Lee, S. Banerjee, P. Sharma, P. Yalagandula, and S. Basu. Bandwidth-aware routing in overlay networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1732–1740, 13-18 2008.

[LG01] Sung-Ju Lee and Mario Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks. In *ICC*, pages 3201–3205. IEEE, 2001.

[MCM09] A.R. Mahlous, B. Chaourar, and M. Mansour. Performance evaluation of max flow multipath protocol with congestion awareness. In *Advanced Information Networking and Applications Workshops, 2009. WAINA '09. International Conference on*, pages 820–825, May 2009.

[MFC08] Ahmed Redha Mahlous, Rod J. Fretwell, and Brahim Chaourar. Mfmp: Max flow multipath routing algorithm. In *Proceedings of the 2008 Second UKSIM European Symposium on Computer Modeling and Simulation*, EMS '08, pages 482–487, Washington, DC, USA, 2008. IEEE Computer Society.

[MLMB01] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John W. Byers. Brite: An approach to universal topology generation. In *MASCOTS*. IEEE Computer Society, 2001.

[MP09] L. Muscariello and D. Perino. Evaluating the performance of multi-path routing and congestion control in presence of network resource management. In *Ultra Modern Telecommunications Workshops, 2009. ICUMT '09. International Conference on*, pages 1–8, October 2009.

[NT99] Wael Noureddine and Fouad Tobagi. Selective back-pressure in switched ethernet lans. In *Proceedings of IEEE GLOBECOM*, pages 1256–1263, 1999.

[NZ01] Srihari Nelakuditi and Zhi-Li Zhang. On selection of paths for multipath routing. In *Proceedings of the 9th International Workshop on Quality of Service*, IWQoS '01, pages 170–186, London, UK, 2001. Springer-Verlag.

[O'H07] John O'Hara. Toward a commodity enterprise middleware. *Queue*, 5(4):48–55, May 2007.

[OIM09] Yasuhiro Ohara, Shinji Imahori, and Rodney Van Meter. Mara: Maximum alternative routing algorithm. In *INFOCOM*, pages 298–306. IEEE, 2009.

[PB03] Peter R. Pietzuch and Sumeer Bhola. Congestion control in a reliable scalable message-oriented middleware. In *Middleware '03: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, pages 202–221, New York, NY, USA, 2003. Springer-Verlag New York, Inc.

[Per05] Colin Perkins. Building adaptive applications: on the need for congestion control. *Proc. SPIE*, 5685:595–602, Apr 2005.

[PMDC03] R. S. Prasad, M. Murray, C. Dovrolis, and K. Claffy. Bandwidth estimation: Metrics, measurement techniques, and tools. *IEEE Network*, 17:27–35, 2003.

[RFH$^+$01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[SKY11] Sushant Sharma, Dimitrios Katramatos, and Dantong Yu. End-to-end network qos via scheduling of flexible resource reservation requests. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, New York, NY, USA, 2011. ACM.

[SL04] Wen-Zhan Song and Xiang-Yang Li. Cbrbrain: provide content based routing service over internet backbone. *Computer Communications and Networks, 2004. ICCCN 2004. Proceedings. 13th International Conference on*, pages 101–106, October 2004.

[SM12] Dr. P.K. Suri and Sumit Mittal. Handling of congestion in cluster computing environment using mobile agent approach. *Global Journal of Computer Science and Technology*, 12(5), 2012.

[SMN$^+$08] H. Subramoni, G. Marsh, S. Narravula, Ping Lai, and D.K. Panda. Design and evaluation of benchmarks for financial applications using advanced message queuing protocol (amqp) over infiniband. In *High Performance Computational Finance, 2008. WHPCF 2008. Workshop on*, pages 1–8, Nov 2008.

[SS12] Xiaping Shi and Dongdong Shi. Cloudqueue: An internet-scale messaging infrastructure based on hadoop. In *Computer Science Education (ICCSE), 2012 7th International Conference on*, pages 335–339, July 2012.

[SU01] B. Subbiah and Z.A. Uzmi. Content aware networking in the internet: issues and challenges. *Communications, 2001. ICC 2001. IEEE International Conference on*, 4:1310–1315, 2001.

[SWW$^+$04] Yan-Tai Shu, Guang-Hong Wang, Lei Wang, Oliver W. W. Yang, and Yong-Jie Fan. Provisioning qos guarantee by multipath routing and reservation in ad hoc networks. *Journal of Computer Science and Technology*, 19(2):128–137, March 2004.

[SYD13] W Sliwinski, I Yastrebov, and A Dworak. Middleware Proxy: A Request-Driven Messaging Broker For High Volume Data Distribution . Technical Report CERN-ACC-2013-0237, CERN, Geneva, Oct 2013.

[UKB02] Guillaume Urvoy-Keller and Ernst Biersack. A multicast congestion control model for overlay networks and its performance. In *Proceedings of 4th International Workshop on Networked Group Communication (NGC)*, pages 141–147, 2002.

[VBB03] Antonino Virgillito, Roberto Beraldi, and Roberto Baldoni. On event routing in content-based publish/subscribe through dynamic networks. In *FTDCS '03: Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, Washington, DC, USA, 2003. IEEE Computer Society.

[WBW11] Jinfu Wang, John Bigham, and Jiayi Wu. Enhance resilience and qos awareness in message oriented middleware for mission critical applications. In *ITNG*, pages 677–682. IEEE Computer Society, 2011.

[WWK$^+$07] Bing Wang, Wei Wei, Jim Kurose, Don Towsley, Krishna R. Pattipati, Zheng Guo, and Zheng Peng. Application-layer multipath data transfer via tcp:

Schemes and performance tradeoffs. *Perform. Eval.*, 64(9-12):965–977, October 2007.

[XJT09] Guo Xin, Zhang Jun, and Zhang Tao. A distributed multipath routing algorithm to minimize congestion. In *Digital Avionics Systems Conference, 2009. DASC '09. IEEE/AIAA 28th*, October 2009.

[YKK+09] Hao Yang, Minkyong Kim, Kyriakos Karenos, Fan Ye, and Hui Lei. Message-oriented middleware with qos awareness. In *Proceedings of the 7th International Joint Conference on Service-Oriented Computing*, ICSOC-ServiceWave '09, pages 331–345, Berlin, Heidelberg, 2009. Springer-Verlag.

[ZDA06] Yong Zhu, Constantinos Dovrolis, and Mostafa Ammar. Dynamic overlay routing based on available bandwidth estimation: A simulation study. *Computer Networks*, 50, 2006.

[ZT10] Azita Zolfaghari and Hassan Taheri. Improving performance of backpressured packet networks by integrating with an end-to-end congestion control algorithm. *Computer Communications*, 33(1):43–53, 2010.

Appendix A

Model Generator

```
#include <limits>
#include <set>
#include <map>
#include <queue>
#include <string>
#include <vector>
#include <fstream>
#include <iostream>
#include <algorithm>
#include <sstream>
#include <math.h>
#include "kpaths.hh"

using namespace std;

map<int, Msg> msgs; /* msg ID -> msg */
vector<tbNode> nodes;
/* note that the path includes the src and dest */
map < int, map < int, vector < vector<int> > > > paths;
double publishers_decimal;
int num_nodes;
int num_msgs;

void set_nodes(const char* topology) {
  ifstream top(topology);
  string line;
  int node_num, node_num2;
  set<int> tmp_nodes;
  map<int, vector<int> > neighbours;
  /* skip over header */
  getline(top, line);
  getline(top, line);
  while (getline(top, line)) {
    stringstream ss(line);
    ss >> node_num;
    tmp_nodes.insert(node_num);
    ss >> node_num2;
    tmp_nodes.insert(node_num2);
    if (node_num != node_num2)
      neighbours[node_num].push_back(node_num2);
  }
  vector<tbNode>::iterator it;
  set<int>::iterator sit;
  for(it = nodes.begin(), sit = tmp_nodes.begin();
      it != nodes.end(), sit != tmp_nodes.end();
      ++it, ++sit) {
    tbNode n;
    /*
     * BRITE nodes are numbered from 0
     * but the sff_out nodes are numbered from 1
     * we use the sff convention
     */
    n.id = *sit;
    /* get random number distributed according to power law with -1.1 exponent */
    n.outgoing_links = neighbours[*sit];
    nodes.push_back(n);
  }
}

void set_pubsub(const char* sff_out, int seed, int buf_size, const char* bandwidth) {
  /* read in the subscribers */
  ifstream sff(sff_out);
  string line;
  int node_num;
  vector<tbNode>::iterator nit;
  int msg_num = 1;
  while (getline(sff, line)) {
    stringstream ss(line);
    while (ss >> node_num) {
      msgs[msg_num].subscribers.push_back(node_num);
      msgs[msg_num].size = 1;
      msgs[msg_num].bunched = false;
    }
    msg_num++;
  }

  vector<tbNode>::iterator dit;
  set<int>::iterator subit;
  ofstream bw(bandwidth);
  vector<int>::iterator n_it;
  for (dit=nodes.begin(); dit != nodes.end(); ++dit) {
    dit->buffer_size = (buf_size * dit->outgoing_links.size());
    for(n_it = dit->outgoing_links.begin(); n_it != dit->outgoing_links.end(); ++n_it) {
      bw << dit->id - 1 << "␣" << *n_it - 1 << "␣" << dit->buffer_size << endl;
    }
  }
  bw.close();
  srand (seed);
  int num_publishers = nodes.size() * publishers_decimal;
  /*
   * note that this vector contains the INDECES of the random publishers, not
   * their actual node numbers (i.e. this starts from 0 whereas nodes are
   * numbered from 1)
   */
  vector<int> rand_publishers;
  int rand_pub;
  for (int i = 0; i < num_publishers; i++) {
    rand_pub = rand() % nodes.size();
    /* make sure we get unique publishers */
    while (find(rand_publishers.begin(), rand_publishers.end(), rand_pub) != rand_publishers.end())
      rand_pub = rand() % nodes.size();
    rand_publishers.push_back(rand_pub);
  }

  /* set each message to be published by a random node */
  map<int, Msg>::iterator mit;
  int n_ind = 0;
  for (mit = msgs.begin(); mit != msgs.end(); ++mit) {
    if (n_ind == rand_publishers.size() - 1)
      n_ind = 0;
    mit->second.publisher = rand_publishers[n_ind] + 1;
    nodes[rand_publishers[n_ind]].publications.push_back(mit->first);
    n_ind++;
  }
  /* set the decimal buffer sizes */
  vector<tbNode>::iterator dit2;
  for (dit2=nodes.begin(); dit2 != nodes.end(); ++dit2) {
    dit2->buffer_size_decimal = 1.0 / dit2->buffer_size;
  }
}

void combine_messages() {
  map<int, Msg>::iterator mit, mit2, mit3;
  for (mit = msgs.begin(); mit != msgs.end(); ++mit) {
    mit3 = ++mit;
    --mit;
    for(mit2 = mit3; mit2 != msgs.end(); ++mit2) {
      if(mit->second.publisher == mit2->second.publisher &&
         mit->second.subscribers == mit2->second.subscribers) {
        mit->second.size++;
        /* the message that is made of smaller ones has bunched = false */
        mit2->second.bunched = true;
      }
    }
  }
}

void get_trees(const char* delays, int req_trees) {
  initialise(delays);
  /*
   * currently when we request 3 trees,
   * the first tree will have the shortest paths for all receivers
   * the 2nd tree will have the 2nd shortest paths for all receivers
   * and so on
   */
  map<int, Msg>::iterator mit;
  vector<int>::iterator sit;
  vector< vector<int> >::iterator self_it;

  for (mit = msgs.begin(); mit != msgs.end(); ++mit) {
    if (!mit->second.bunched && mit->second.subscribers.size() > 0) {
      mit->second.trees.resize(req_trees);
      for (sit = mit->second.subscribers.begin(); sit != mit->second.subscribers.end(); ++sit) {
```

```
/*
 * if the publisher and subscriber are the same node
 * this node is already part of the tree
 */
if (mit->second.publisher != *sit) {
  bool is_calculated = false;
  bool is_contained = false;
  /* if this path is already stored then just copy it */
  if (paths[mit->second.publisher][*sit].size() != 0) {
    for (int i = 0; i < req_trees; i++)
      mit->second.trees[i] = paths[mit->second.publisher][*sit][i];
    is_calculated = true;
  }
   else {
    /* else check if this path is contained */
    /*
     * look in all paths starting from this source to any dest
     * if any k paths to any dest contain this dest
     *   truncate those paths so that they end at this dest
     */
    int index;
    vector<int> positions;
    map< int, vector < vector<int> > >::iterator i;
    vector <vector <int > >::iterator j;
    vector<int >::iterator p_i;
    for (i = paths[mit->second.publisher].begin(); i != paths[mit->second.publisher].end(); i++) {
      positions.clear();
      for (j = i->second.begin(); j != i->second.end(); j++) {
        index = distance(j->begin(), find(j->begin(), j->end(), *sit));
        if (index < j->size()) {
          positions.push_back(index);
          if (positions.size() == req_trees) {
            int k_ctr = 0;
            paths[mit->second.publisher][*sit].resize(req_trees);
            for(p_i = positions.begin(), j = i -> second.begin();
                p_i != positions.end(), j != i -> second.end(); ++p_i, ++j) {
              paths[mit->second.publisher][*sit][k_ctr].assign(j->begin(), j->begin() + *p_i + 1);
              k_ctr++;
            }
            is_contained = true;
            goto contained;
          }
        }
      }
    }
  }
  contained:
  if (!is_calculated && !is_contained) {
    get_paths(&msgs, &(mit->first), mit->second.publisher - 1, *sit - 1, req_trees, &paths, &nodes);
  }
}
/* else we are both subscriber and publisher so we add ourselves to our trees */
else {
  for(self_it = msgs[mit->first].trees.begin(); self_it != msgs[mit->first].trees.end(); ++self_it) {
    self_it->push_back(*sit);
  }
  if(msgs[mit->first].trees.size() == 0) {
    vector<int> tmp;
```

```
            msgs[mit->first].trees.push_back(tmp);
            msgs[mit->first].trees[0].push_back(*sit);
          }
        }
      }
    }
  }
  /* fill out trees */
  vector<int>::iterator s_it;
  for (mit = msgs.begin(); mit != msgs.end(); ++mit) {
    if (!mit->second.bunched && mit->second.subscribers.size() > 0) {
      for (s_it = mit->second.subscribers.begin(); s_it != mit->second.subscribers.end(); ++s_it) {
        if(*s_it != mit->second.publisher) {
          for (int path = 0; path < paths[mit->second.publisher][*s_it].size(); ++path ) {
            if(paths[mit->second.publisher][*s_it][path].size() == 0) {
              paths[mit->second.publisher][*s_it][path] = paths[mit->second.publisher][*s_it][path - 1];
              mit->second.trees[path] = paths[mit->second.publisher][*s_it][path - 1];
            }
          }
        }
      }
    }
  }
}

bool generate_suboptimal_model(const char * mu_model, const char* lb_model) {
  ofstream fout(mu_model);
  ofstream fout2(lb_model);
  if(!fout.is_open() || !fout2.is_open()) {
    cerr << ”Error:_unable_to_create_optimisation_model_files.” << endl;
    return false;
  }

  map<int, Msg>::iterator mit;
  vector< vector <int> >::iterator tit;
  vector<tbNode>::iterator nit;

  /* remove duplicates from trees */
  for(mit = msgs.begin(); mit != msgs.end(); ++mit) {
    for (tit = mit->second.trees.begin(); tit != mit->second.trees.end(); ++tit) {
      std::sort(tit->begin(), tit->end());
      tit->erase(std::unique(tit->begin(), tit->end()), tit->end());
    }
  }

  int tree_usage = 0; /* size of bunched m * number of nodes in tree of m */
  int t_ctr;
  fout << ”Minimize”;
  fout2 << ”Minimize_y”;
  for (mit = msgs.begin(); mit != msgs.end(); ++mit) {
    if (!(mit->second.bunched) && mit->second.subscribers.size() > 0) {
      t_ctr = 1;
      for (tit = mit->second.trees.begin(); tit != mit->second.trees.end(); ++tit) {
        tree_usage = mit->second.size * tit->size();
        fout << ”_+_” << tree_usage << ”_x” << ”_” << mit->first << ”_” << t_ctr;
        t_ctr++;
      }
    }
```

```
}

fout << endl << endl << "Subject␣To" << endl << endl;
fout2 << endl << endl << "Subject␣To" << endl << endl;

for (mit = msgs.begin(); mit != msgs.end(); ++mit) {
  if (!(mit->second.bunched) && mit->second.subscribers.size() > 0) {
    t_ctr = 1;
    for (tit = mit->second.trees.begin(); tit != mit->second.trees.end(); ++tit) {
      fout << "␣+␣x␣" << mit->first << "␣" << t_ctr;
      fout2 << "␣+␣x␣" << mit->first << "␣" << t_ctr;
      t_ctr++;
    }
    fout << "␣>=␣" << mit->second.size << endl;
    fout2 << "␣>=␣" << mit->second.size << endl;
  }
}

fout << endl;
fout2 << endl;

bool is_in_tree;
for (nit = nodes.begin(); nit != nodes.end(); ++nit) {
  is_in_tree = false;
  for (mit = msgs.begin(); mit != msgs.end(); ++mit) {
    if (!(mit->second.bunched) && mit->second.subscribers.size() > 0) {
      t_ctr = 1;
      for (tit = mit->second.trees.begin(); tit != mit->second.trees.end(); ++tit) {
        if(find(tit->begin(), tit->end(), nit->id) != tit->end()) {
          fout << "␣+␣" << "x␣" << mit->first << "␣" << t_ctr;
          fout2 << "␣+␣" << "x␣" << mit->first << "␣" << t_ctr;
          is_in_tree = true;
        }
        t_ctr++;
      }
    }
  }
  if (is_in_tree) {
    fout << "␣<=␣" << nit->buffer_size << endl;
    fout2 << "␣<=␣" << nit->buffer_size << endl;
  }
}

/* constraint only for load balancing goal */
fout2 << endl;
bool is_in_tree2;
for (nit = nodes.begin(); nit != nodes.end(); ++nit) {
  is_in_tree2 = false;
  for (mit = msgs.begin(); mit != msgs.end(); ++mit) {
    if (!(mit->second.bunched) && mit->second.subscribers.size() > 0) {
      t_ctr = 1;
      for (tit = mit->second.trees.begin(); tit !=
      mit->second.trees.        end(); ++tit) {
        if(find(tit->begin(), tit->end(), nit->id) != tit->end()) {
          fout2 << "␣+␣" << "x␣" << mit->first << "␣" << t_ctr;
          is_in_tree2 = true;
        }
        t_ctr++;
        }
      }
    }
    if (is_in_tree2)
      fout2 << "␣-␣y␣<=␣0" << endl;
  }

  fout << endl << endl << "Bounds" << endl;;
  fout2 << endl << endl << "Bounds" << endl;;

  for (mit = msgs.begin(); mit != msgs.end(); ++mit) {
    if (!(mit->second.bunched) && mit->second.subscribers.size() > 0) {
      t_ctr = 1;
      for (tit = mit->second.trees.begin(); tit != mit->second.trees.end(); ++tit) {
        fout << "0␣<=␣x_" << mit->first << "_" << t_ctr << "␣<=␣" << mit->second.size << endl;
        fout2 << "0␣<=␣x_" << mit->first << "_" << t_ctr << "␣<=␣" << mit->second.size << endl;
        t_ctr++;
      }
    }
  }
  vector<int>::iterator s_it;
  cout << "begin␣paths␣output␣\n\n\n";
  for (mit = msgs.begin(); mit != msgs.end(); ++mit) {
    if (!mit->second.bunched && mit->second.subscribers.size() > 0) {
      cout << "message␣" << mit->first << endl;
      for (s_it = mit->second.subscribers.begin(); s_it != mit->second.subscribers.end(); ++s_it) {
        for (int path = 0; path < paths[mit->second.publisher][*s_it].size(); ++path ) {
          cout << "tree␣" << path << endl;
          for(int hop = 0; hop < paths[mit->second.publisher][*s_it][path].size(); ++hop)
            cout << paths[mit->second.publisher][*s_it][path][hop] - 1 << "␣";
          cout << endl;
        }
      }
    }
  }
  fout << endl << "End" << endl;
  fout2 << endl << "End" << endl;

  return true;
}

int main(int argc, char** argv) {
  if (argc != 16) {
    cout << "Usage\n\t" << argv[0] << "␣<topology_loops>␣<sff_out>␣"
            "<number_of_trees>␣<so_model_mu>␣<topology_no_loops>␣"
            "<so_model_lb>␣<o_model_mu>␣<o_model_lb>␣<seed>␣"
            "<publishers_decimal>␣<num_nodes>␣<num_msgs>␣"
            "<trees_out>␣<buffer_size>␣<assigned_bandwidth>" << endl;
    return 1;
  }

  publishers_decimal = atof(argv[10]);
  num_nodes = atoi(argv[11]);
  num_msgs = atoi(argv[12]);
  set_nodes(argv[1]);
  set_pubsub(argv[2], atoi(argv[9]), atoi(argv[14]), argv[15]);
  combine_messages();
  get_trees(argv[5], atoi(argv[3]));
  bool is_success = generate_suboptimal_model(argv[4], argv[6]);

  map<int, Msg>::iterator m_it;
  vector< vector<int> >::iterator ts_it;
  vector<int>::iterator t_it;
  ofstream fout(argv[13]);
  int t_ctr = 1;
  for(m_it = msgs.begin(); m_it != msgs.end(); ++m_it) {
    t_ctr = 1;
    for(ts_it = m_it->second.trees.begin(); ts_it != m_it->second.trees.end(); ++ts_it) {
      fout << m_it->first << "_" << t_ctr << "␣";
      fout << ts_it->size() << endl;
      t_ctr++;
    }
  }
  return is_success ? EXIT_SUCCESS : EXIT_FAILURE;
}
```