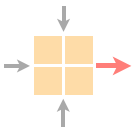
ETH zürich



Networked Systems
ETH Zürich — seit 2015

Interactive OSPF Visualization

Bachelor Thesis

Author: Valentin Jenny

Tutor: Tibor Schneider

Supervisor: Prof. Dr. Laurent Vanbever

October 2023 to January 2024

Abstract

Internet routing is an important concept in today's day and age, however, it is quite difficult to understand for a variety of reasons, making it an interesting field to simulate. Thus we extend an existing routing simulator with a more detailed version of the OSPF protocol, to offer insight into the workings of the protocol for teaching and research. We justify the inclusion and exclusion of certain features based on how much they serve those two purposes. We find that introducing additional accuracy to the simulation adds to the runtime of it. Ultimately, we create a simulator that more accurately models internal routing procedures.

Contents

Chapter 1

Introduction

Here we will give the motivation for this thesis, as well as breaking down the goals we mean to achieve.

1.1 Motivation

Routing protocols are difficult to understand. Not only do they involve intricate computation, but they are distributed by nature. We believe that the best way to learn about routing protocols is hands-on: to set up the routing protocol in a network, modify the configuration, and see how this impacts the network state.

However, most people do not just have access to real network hardware and it is suboptimal for learning anyway. Real hardware has many problems with observation, as one can not easily pause the convergence process, and setting up physical hardware for observation is just more difficult in general. E.g. it is much easier to add another simulated router to the network than having to physically hook up additional hardware.

As such, these networks and protocols are interesting to simulate, both to investigate their behavior under certain conditions and as a tool for teaching about them. The latter especially profits a lot from being able to visualize the processes in the network concisely.

OSPF is one of the most common link-state protocols used for interior routing. While the initial simulation does include a model of OSPF, it does not simulate any actual messages being passed between routers.

We extend this pre-existing simulation with a more accurate version of OSPF which more closely models the communication between routers. However, we still wish to make appropriate abstractions where possible, to keep the complexity as low as possible, while still fulfilling our needs.

The most accurate simulation would of course model every last bit of the protocol down to the exact data structures. However, this comes with several trade-offs.

The simplest one is the effort on implementation. A simpler model may yield all the same results we are interested in while being significantly easier to implement.

Another consideration is the performance of the simulator, modeling things we are not ultimately interested in wastes computing time.

Of course, abstracting away behavior that is interesting to observe goes against the basic idea of the simulator. So a proper middle ground must be chosen.

1.2 Task and Goals

The first goal is to find an appropriate level of abstraction for our simulator. This first requires a solid understanding of the protocol itself.

The primary task then was to implement our chosen model of OSPF for the already existing simulation. That requires us to understand the connection points already exposed by the framework.

We achieve most of our goals in this step, however, OSPF areas have to be dropped from our scope.

Lastly, we work on visualizing the whole process. Here we unfortunately only got to the ideation step, we did not have enough time to implement our ideas.

1.3 Overview

Section 2 describes the most relevant parts of the OSPF protocol as described in RFC 2328 [2], Section 3 presents details on the exact implementation and some of the thought processes behind where we chose to introduce layers of abstraction, Section 4 contains some insights on performance, and finally Section 5 gives a non-exhaustive view of possible next steps.

Chapter 2

Background

In this chapter, we will first give an insight into the simulator we extend. Then we will give a rundown on the concepts of the OSPF (Open Shortest Path First) protocol required for this thesis.

2.1 The initial simulator

The simulator was created during Tibor Schneider's master's thesis [3]. It simulates a network of routers, using BGP (Border Gateway Protocol) as its Exterior Gateway Protocol and OSPF as its IGP (Interior Gateway Protocol).

2.1.1 BGP

BGP ensures that all internal routers of the AS (Autonomous System) are aware of external destinations, and which internal router(s) they need to reach on route to the destination. BGP achieves this by sending messages between the internal routers, advertising, or retracting advertisements to certain routes.

2.1.2 IGP

The IGP meanwhile needs to ensure that all AS internal routers know how to reach all other internal routers. Some commonly used ones include IS-IS, EIGRP, and OSPF.

The Interior Gateway Protocol used for the simulator is OSPF, however, any message passing has been abstracted from the model. Instead, the computation is done for the entire (internal) network instantaneously. This is a reasonable assumption as IGPs generally converge significantly faster than BGP.

2.1.3 The Message Queue

The messages passed between routers are handled via a simple queue, resolving one after another. There is no notion of some absolute timestep at which a certain message would arrive at a router.

The queue itself can be modified to model certain behavior (e.g. by allowing for random reordering of messages in the queue.) but it will still always be a somewhat relative notion of time. We will see some of the consequences of that later in Section 3.3.2.

2.2 OSPF

As an Interior Gateway Protocol, the goal of OSPF is to make sure that all routers in an AS are aware of their next hop for any relevant internal destinations.

As a link-state protocol, OSPF achieves this by ensuring that each router knows the entire state of the network at all times (or at least as quickly as updates can travel through the network). To this effect, each router keeps a Link-State Database where it stores everything it currently knows about the links of the network.

2.2.1 The notion of Neighbor and Adjacency

OSPF uses Hello-Packets to discover viable neighbors, as well as to learn some basic information about those neighbors. These packets are periodically sent out all interfaces of a router. If a router receives a Hello-Packet that mentions itself, it knows this is now a fully 2-way connection and is now considered a full Neighbor.

Neighbors can be further brought up to an Adjacency through the Database Exchange Process (See Section 2.2.3). In the case of point-to-point networks, this is always done. (For other network types, the decision-making on whether to bring up an Adjacency is significantly more involved and requires concepts not further relevant to our simulation. See [2] Chapter 7. & Section 10.4)

Only Adjacencies exchange data about the network via OSPF.

As for our purposes 'Adjacency' and 'Neighbor' in the OSPF sense are essentially synonymous we will be using Adjacency whenever we refer to the OSPF concept, leaving 'neighbor' free for its more generic use.

2.2.2 The Link-State Database

The link-state database of a router gathers the information about the network that it receives from its adjacencies. This information is distributed in the form of link-state advertisements (LSAs). RFC 2328 [2] defines five different LSAs.

These different types of LSAs are used in different contexts to reduce the amount of information that needs to be seeded throughout the network, allowing OSPF to scale to larger networks.

Each LSA has an associated sequence number and age. The sequence number is a sort of epoch, which is used by routers when determining which information is more recent and should be kept if two LSAs describing the same links need to be compared. The age field tracks how long ago this LSA was initially issued. Usually, a router would delete LSAs above a certain age. However, since our simulation uses a very simplified timing model, the age of an LSA is only relevant for flushing unwanted advertisements from the network (See Section 3.1.3).

The next sections will give a high-level overview of the LSAs relevant to our simulation.

Router LSAs

Each router issues a single Router LSA, in which it describes all of its outgoing links to other routers. Router LSAs are only shared within a single area.

Summary LSAs

OSPF allows for the creation of areas. Across the borders of such areas, information is shared only in a simplified form, which gives OSPF the ability to scale better to larger networks.

Summary LSAs are this simplified information and are issued by area border routers. They condense the information of Router LSAs for propagation across area borders.

AS External LSAs

These are issued by AS boundary routers and describe links leading outside the autonomous system. Unlike Router LSAs, these are flooded through the entire AS without caring about OSPF areas.

2.2.3 Ensuring two adjacent routers are always synchronized

As a link-state protocol, it is vitally important that OSPF can ensure at all times that two adjacent routers are fully updated. OSPF achieves this in two steps.

The Database Exchange Process

When two previously not adjacent routers wish to raise an Adjacency, they start the Database Exchange Process. Here both routers first describe their respective Link-State-Database to each other, by sending the headers of all local LSAs to their neighbor.

The header of an LSA describes which link (or in the case of the router LSA, which issuing router) it contains information for, as well as the sequence number which is used to determine the LSA's relative recency. If the adjacency holds a more recent version of any LSAs, or entirely new LSAs the router will request those LSAs.

If the Adjacency does request some subset of LSAs, they are placed on that Adjacency's Link-state Retransmission List and sent to the Adjacency.

Figure 2.1 shows a visualization of this process. Before the process can begin the two routers need to designate a leader and a follower. This is done by designating the router with a lower router-id as leader.

The leader initiates the process, and the follower will only ever respond to a message from the leader (at least until LSAs are placed onto the retransmission list, at which point the regular mechanism for transmitting advertisements takes over. See Section 2.2.3)

The Link-state Retransmission List

The Link-state Retransmission List ensures that an LSA required by an Adjacency will arrive there.

Each router has one such list for each of its Adjacencies. Any LSA sent to an Adjacency is placed on that Adjacency's Retransmission List.

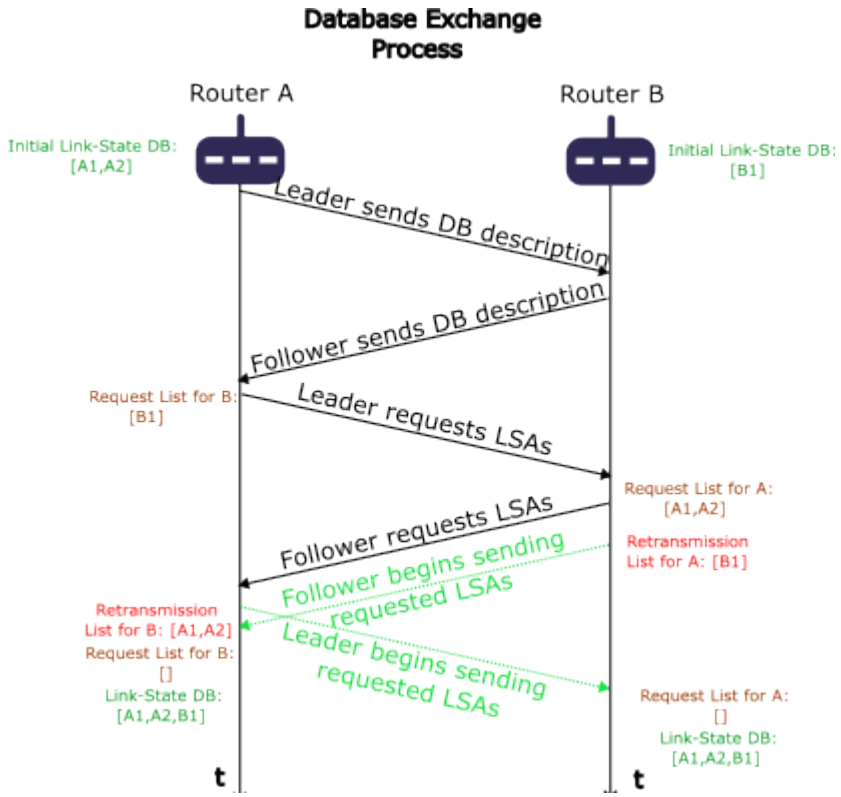
The LSAs on this list are periodically retransmitted until the Adjacency acknowledges them. Each individual LSA must be acknowledged for it to be taken off the list, however, this may not necessarily happen through an explicit acknowledgment as there are a few different ways through which an LSA may be acknowledged (See Section 2.2.4).

2.2.4 The Flooding Procedure

The flooding procedure is where a lot of the actual decision-making of the protocol happens. It decides what should be done with any incoming LSAs. What follows here is a rough overview of the higher-level thought processes behind it.

- If the incoming LSA is more recent than our local copy:
  - Install the new LSA in our local database

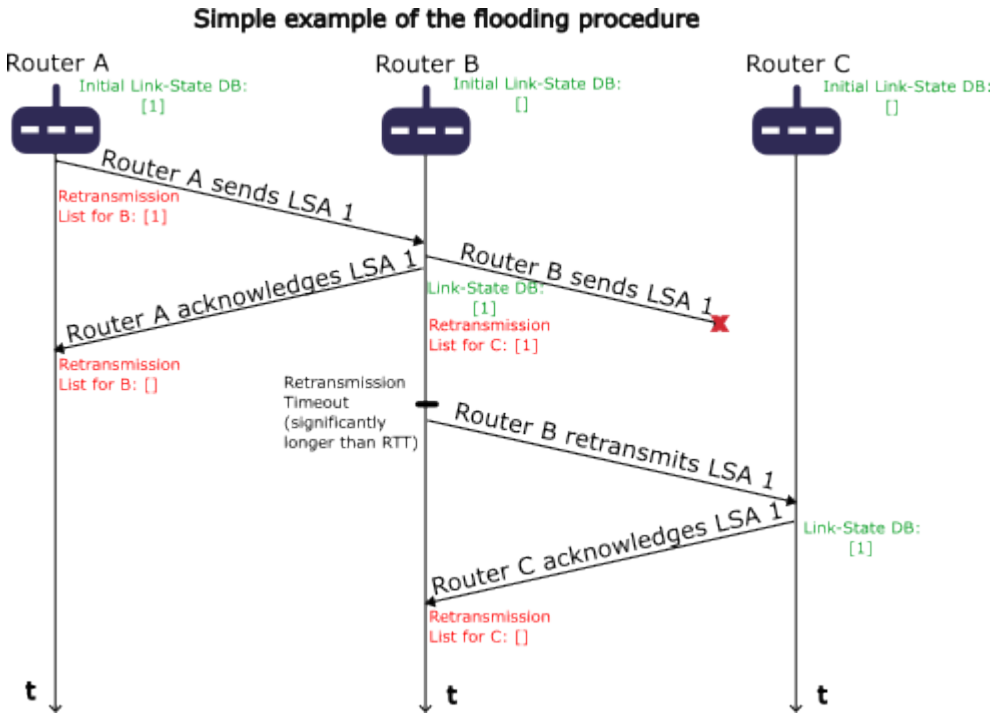
Figure 2.1: Two routers going through the exchange process, assuming the entire description and all requests fit into one message each



  - Acknowledge it
  - Then send it to our other adjacencies

- If the incoming LSA is equally recent as our local copy:
  - If this LSA is on this adjacency's retransmission list, this is treated as an acknowledgment
  - Otherwise simply acknowledge and discard it
- If the incoming LSA is less recent than our local copy:
  - Send back our local copy without acknowledging the received LSA

Figure 2.2: Example of the flooding procedure where one LSA is dropped



We will cover a few more special cases in Section 3.1.3.

Chapter 3

Design

In this chapter, we will discuss specific design choices and how they relate to our goals of building a simulator for teaching and research.

3.1 Maintaining the Link-State Database

The most pivotal part of the protocol is making sure that all routers agree on the state of the network. In the previous model, this was simply assumed as given, and thus the biggest chunk of extending the simulation is implementing the scheme for passing messages between the routers.

Almost all messages, aside from the Hello Packet which we disregard, are only passed between adjacencies.

3.1.1 Bringing up an Adjacency

The transition between what constitutes a dead link and an OSPF Adjacency is among the big cut-off points where our simulation stops modeling the protocol as closely. And so we find it prudent to expand on it first.

Usually, there would be some lower-level processes that give information about the links that are up and available to send messages across. From there, usually OSPF discovery would work via Hello Packets. However, we abstract both of those away, since they deal with the discovery of the underlying network topology, rather than the communication between routers to build the individual information into a cohesive map of the network. The Hello-Packet also has some further complications regarding our timing model, see Section 3.3.2.

Thus we start up the Database Exchange Process whenever a **bi-directional** link between two internal routers has been defined in the network. Since the hello packet ensures this bi-directionality, we need to ensure it on our own here.

The Database Exchange Process ensures that routers that enter the network are brought up to date. To that effect, the two newly connected routers send each other a snapshot of their current databases. Both routers then request the advertisements they require from their new adjacency, expecting to receive them soon. (See Figure 2.1 for an example)

This is done to minimize the amount of actual data that needs to be exchanged between the routers.

There are some simplifications to the process we can make here, see Section 3.1.2

During this process, the router already sends new advertisements that are relevant to the Adjacency to it. However, the link to the Adjacency is not yet described in the Router LSA until all advertisements requested from the Adjacency initially are received.

Once an adjacency is brought up between two routers, they will now be able to communicate via certain predefined packets.

3.1.2 Packets

Of the five packets defined in RFC 2328 [2] four are implemented. The hello packet is omitted (See Section 3.3.2)

Linkstate Update Packet

This packet is what facilitates the sharing of Link-State Advertisements between routers. It is generally the most expensive to send, both in terms of being the packet that contains the most information and also the one that prompts the most involved response by the receiving router (See Section 3.1.3)

The router will usually place any sent LSA on the retransmission list of the corresponding adjacency and will thus expect it to be acknowledged sometime in the future (unless of course the adjacency is torn down either in part or entirely).

Even though we assume that no packets are dropped or corrupted, explicit acknowledgment is required when flooding a max-age advertisement to flush it from the network. Such an advertisement needs to be held by the router until all of its adjacencies have acknowledged it, at which point it can be sure it will no longer be required and can delete it, potentially making space for a newly issued LSA describing the same link.

Linkstate Acknowledgement Packet

Routers use this packet to acknowledge incoming LSAs. When a router receives such a package it knows that the corresponding LSAs need not be re-transmitted and it can safely remove them from the adjacency's re-transmission list. They are however not the only way an LSA can be acknowledged.

Sometimes Linkstate Update Packets are treated as acknowledgments, a so-called implied acknowledgment. The most likely situation where this happens is when two routers send each other the same LSA at the same time (i.e. both routers decide the neighbor needs this LSA before either one of them receives it from said neighbor). This saves on link traffic for unnecessary acknowledgments.

Database Description and Link-State Request Packet

These packets are used during the Database Exchange Process to kick-start the synchronization of the databases. During the process, a Leader-Follower relationship is established between the two routers. This serves mostly to make sure that in the event a packet is dropped, it is definitely retransmitted until it is acknowledged by the other party. That means the process is vastly simplified in our implementation since we both assume that the entire database can be described in a single message and that no messages are ever dropped.

The Database Description packet is used to describe the current Link-State Database to the Adjacency, whereas the Link-State Request packet is used to request some subset of advertisements from our adjacency (See Section 2.2.3).

3.1.3 The Flooding Procedure

With all the pieces in place that are used to describe the network, we will now cover the individual decision-making that each router makes such that the network converges to its correct state.

As the basics have already been covered in Section 2.2.4 we will talk about the more edge case decision-making and why we choose to include it or not.

Once again this procedure can be significantly simplified due to our assumptions.

Handling LSAs that are currently being flushed from the network

Sometimes we wish to completely remove an advertisement from the network. This happens when the sequence number of an LSA reaches its maximum value and needs to be wrapped. In that situation, we flood the network with our LSA with maximum sequence number and age. This LSA will be accepted by every router, then transmitted to all of that router's adjacencies (except the one it received it from), and then finally deleted from that router's database. This propagates throughout the entire network until the LSA has been entirely removed, i.e. flushed.

The router who initially issued the wrapping LSA will wait until all Adjacencies have acknowledged the LSA with maximum age and sequence number, before issuing a new LSA with the minimum sequence number.

Special consideration needs to be given in the case where such an LSA arrives at a router that does not have the LSA that is being flushed to begin with. The router must first check whether it is fully synchronized with all of its Adjacencies (i.e. if it is currently in the Database Exchange Process with any of them). If it is fully synchronized with all Adjacencies, the incoming LSA is simply acknowledged but not taken into the local database. Otherwise, the Adjacencies that are not fully synchronized yet might require this LSA and it must be sent to them.

Conversely, should a new LSA come in for which a router has a local copy that is maximum age, which it has not discarded yet because not all Adjacencies have acknowledged it, the incoming LSA is simply discarded without acknowledging. It expects to receive it later again, at which point hopefully its Adjacencies have acknowledged it so the router can accept the new version of this LSA.

Handling self-issued LSAs

One last edge case that needs to be covered is when a router receives an LSA that is ostensibly issued by itself, but the received LSA is more recent than its local copy. This usually happens when a router is restarted wiping its database in the process. When the router then is reconnected before its issued advertisements have time to age out of the network (in our case that will always happen as advertisements do not naturally age) it will receive the version of its LSA that was issued before the restart.

In that case, the local LSA needs to be re-flooded with a "fast-forwarded" sequence number if it still exists after the restart, or flooded with maximum age and sequence number to flush the entry if it does not exist.

Error Correction

Usually, the Flooding Procedure catches errors in the Database Exchange Process where our Adjacency sends us an advertisement it requested from us earlier. This signals that some sort of mistake happened during the initial exchange of databases, and thus the entire process needs to be started over.

It would also sanitize the type of LSA coming in, dropping it if it is not one of the 5 defined in the protocol.

Lastly, it also checks the checksum of any incoming packets to see whether they have been corrupted.

Since our simulator does not model packets being corrupted or outright wrong packets being sent we are not interested in these parts of the procedure.

3.1.4 Retransmitting in a simple timing model

As previously discussed our timing model does not track time in an absolute sense instead, it simply queues up events and works through them one at a time. This leaves us with a bit of a problem when it comes to an event that is supposed to trigger on a timeout, like retransmitting advertisements that have not been acknowledged yet.

We choose here to simply check all retransmission lists whenever our event queue is empty, and simply pick one router at random that should start its retransmissions. This approach is reasonable, as the time it takes for a network to converge should be significantly lower than the timer that governs retransmission.

Section 3.3.3 goes into more detail on why we require retransmission in our abstraction in the first place.

3.2 Calculating the routing table

Now that we've ensured that our information about the network is up to date, we want to piece the advertisements together to build our routing table.

Ultimately what we need to hand off the BGP part of our simulation is a list of all reachable routers with their corresponding next hop(s) and the total cost to get there.

3.2.1 Constructing the current network

In the original OSPF protocol, the routers only construct the shortest path tree out of its LSAs for further computation of the next hops.

For simplicity, however, we haveve decided to just compute the entire network graph instead. We will see later that this has a comparatively low impact on the performance for our purposes, but greatly simplifies the process of constructing our next hops.

This is done by simply iterating over all LSAs the router currently has access to and adding all edges described in those advertisements to our copy of the network. This approach once again has the big advantage of being extremely simple, although extra performance could be gained by choosing a smarter approach that only adjusts the graph in places where we have received an LSA that would cause a change.

3.2.2 Computing the next hop

The exact details of the next hop calculation are some of the most complicated parts of the OSPF protocol and since those specifics are not really relevant to see the evolution of our network as it converges, we decided to implement a much simpler, if less performant scheme.

We begin by first computing the all-pairs shortest paths for the local copy of our network via the Floyd-Warshall algorithm. This then allows us for each router in our network to check all of

our neighbors' costs to arrive at our destination. If we then also consider the weight to reach the neighbor itself, we can easily compare and find all of the lowest cost next hops.

This extra step is required to determine which of our neighbors is our nexthop, since we need that information along with the total cost to get to our destination.

The Floyd-Warshall algorithm greatly dominates this process so this algorithm is bounded by $\mathcal{O}(V^3)$, where V is the number of routers this router knows about (Checking each of our neighbors is merely linear in the number of our direct neighbors). However, in the worst case, this algorithm is called every time new information about any edge in the network is acquired, on each internal router. We will see the consequences of that in Chapter 4.

3.3 Protocol Abstraction

As with any model ours includes a few simplifying assumptions. Usually, we make these assumptions because whatever they simplify is not interesting to us, at least compared to the cost of implementing it fully.

3.3.1 Non Point-to-Point networks

A large part of the OSPF protocol deals with network architectures that are not simple Point-to-Point networks. However, as our simulation only models Point-to-Point networks, we can disregard all that. A non-exhaustive list of concepts that need not be implemented:

- Anything to do with Designated Routers (See [2] Section 7.3)
- Network-LSAs (See [2] Section 12.4.2)
- Decision-making on whether to bring up a neighbor relationship to an adjacency (See Section 2.2.1)

3.3.2 Timing

Our simulation adopts a significantly simplified timing model, where events happen in sequential order, without any notion of absolute time passing. This has consequences for a few mechanics of the OSPF protocol since they rely on some absolute measure of time.

The Hello-Packet

If a certain timeout is reached without a Hello-Packet being received from a neighbor, the connection is treated as down.

Since the Hello-Packet mostly deals with discovering changes in the network topology and we are mostly interested in how the network behaves after changes, we deemed this particular part of the protocol not interesting enough to directly implement.

Age of LSAs

LSAs usually have a field where they track how much time has passed since they were initially issued. Above a certain age, the advertisement is considered invalid and will be flushed from the database. However, we lack any model for discreet timesteps and as such do not naturally age the advertisements.

However, one application of the age field is used in the simulation. The OSPF protocol uses premature aging to flush out unwanted advertisements from the network. This happens in a few different situations:

- When an AS external link is torn down, flushing out the LSA from the network is more desirable than simply advertising it as infinite link weight, as otherwise an arbitrarily large amount of these advertisements could exist.
- When a router is reset (clearing its local Link-State Database in the process) and then receives one of its previously issued advertisements it no longer wishes to issue.
- When an LSA's sequence number approaches its maximum (per the protocol the number is stored in a 32-bit signed integer) it needs to first be flushed from the network before the LSA with a wrapped sequence number can be reissued.

3.3.3 Error Correction

Since the OSPF protocol does not use TCP or similar to send its messages a lot of effort is spent on sanitizing incoming packets and checking that packets have not been reordered.

By and large, all these error-correcting measures are of little interest to us. The effects caused by these are purely transient, and since the entire point of them is to end up in the correct state eventually, they are not relevant to the actual process of convergence beyond causing some delays.

There is however one such error-correcting measure we implement.

The Link-State Retransmission List

Even though it is theoretically unnecessary if we assume all our messages arrive correctly, there are a few cases where we deliberately choose not to acknowledge an advertisement we are not ready for quite yet and simply rely on the re-transmission to receive it later.

It is also required for a few applications where we need to know whether a certain packet has already been acknowledged, such as when issuing an LSA with its sequence number wrapped, after previously flushing out the wrapping version.

Chapter 4

Evaluation

Here we will discuss some of our findings in regards to performance.

4.1 Scaling the network

Samples were taken by populating various topologies from the internet topology zoo [1] with uniform linkweights. Time measured is until the network is once again fully converged. 30 iterations were run and we report the median value, such as to diminish the impact of random busyness of the CPU or similar.

Figure 4.1 shows the slowest execution time among topologies for each number of internal routers. Figure 4.2 shows the same for each total number of routers. As we can see, while there is a trend of increasing number of routers meaning an increasing execution time, it is not nearly as clear as the trend in figure 4.3, showing the execution time by number of edges.

When looking at the number of edges, the most extreme outliers are mostly topologies that feature a large number of edges, but a comparatively very small number of internal routers.

Figure 4.1: Execution time by internal routers

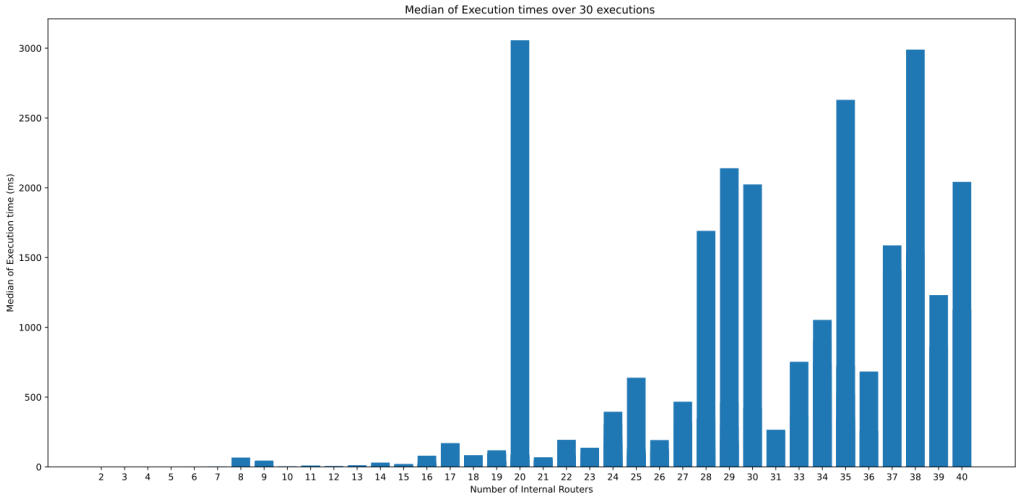
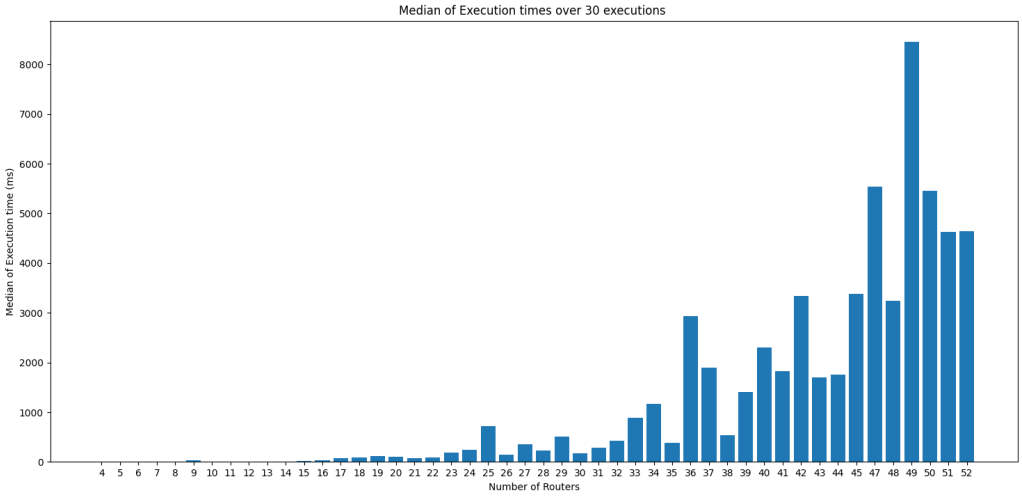


Figure 4.2: Execution time by total routers



4.2 Profiling

Figure 4.4 shows the flame graph for populating the Internode topology with uniform link weights 30 times. The Internode topology features 66 routers, 20 of which are internal routers and 46 external routers, and 77 edges, 31 of which are connecting two internal routers. It was chosen because it represents a nice mid-sized topology with plenty of edges and routers.

As we can very clearly see, around 90% of the execution time is spent in calculating the next hop, in particular, running the Floyd-Warshall algorithm to find the all-pairs shortest paths. This is somewhat surprising since that algorithm scales with $\mathcal{O}(V^3)$, we would expect in the previous section to see a bigger trend when sorting by the number of routers rather than the number of edges.

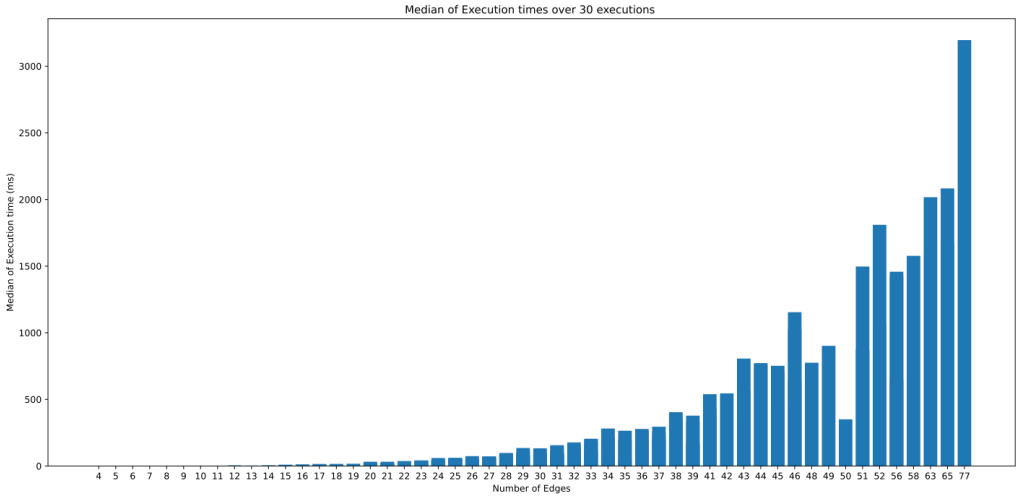
We suspect that since potentially every edge can cause a recalculation of the nexthop in every single router, the rate at which we are calling the nexthop calculation has a potentially bigger impact on our performance than the raw complexity of the algorithm.

Further investigation into the data yields some interesting patterns. In Table 4.1 we see that Internode does have a significantly higher edge count, and also calls the nexthop calculation more often. However, the difference in execution time that we see in Figure 4.1 cannot be fully explained by this difference in calls alone. As Internode also has significantly more total routers, the complexity of our algorithm still seems relevant.

Table 4.2 looks at the outlier from Figure 4.2. Here we can observe the impact of the additional edges quite clearly. Most of the extra computation time stems from the additional calls to the function.

We conclude from this that if we want to increase our performance, we should not only look to reduce the complexity of our nexthop calculation but also reduce the number of times it is called.

Figure 4.3: Execution time by total edges



Topology Name	number of nexthop recalculations	total routers	internal routers	edges
Widejpn	1114	30	19	33
Internode	3678	66	20	77
Vinaren	1108	25	21	26

Table 4.1: Extra Data from the topologies around the 20 internal router outlier seen in Figure 4.1

Topology Name	number of nexthop recalculations	total routers	internal routers	edges
HurricaneElectric	3324	24	24	37
AttMpls	8977	25	25	56
Sunet	1883	26	26	32

Table 4.2: Extra Data from the topologies around the 25 total router outlier seen in Figure 4.2

Figure 4.4: Flamegraph of populating the Internode topology[1] with uniform link weight 1, 30 times

Chapter 5

Outlook

This chapter deals with further features that would be desirable to implement, but for which we lacked the time to do so.

5.1 OSPF Areas

As chapter 4 shows, OSPF's overhead scales quite badly with larger networks. The protocol offers a solution for that in the form of areas. These areas allow to partition the network so less information needs to be flooded through the whole network. This also means that the routing tables need to be calculated less often, as a change in an area does not necessarily lead to a change in the Link-State Databases in other areas.

This would also include properly implementing AS External LSAs, as external links are currently just described in the Router LSA of the border router. External LSAs become more relevant once areas are introduced, since unlike Router LSAs they are flooded across area borders.

Unfortunately, due to time constraints, we could not include areas in our simulation. This would be a natural next step to further extend the simulation, especially as it is also relevant for demonstrating certain behavior in the network that would be interesting to observe.

5.2 Visualizing OSPF

This was also one of the initial goals for which we ultimately did not have enough time.

For teaching purposes especially, being able to concisely visualize the current state of the Link-State Databases would be very advantageous. The biggest hurdle there conceptually would be how one would go about presenting this information nicely. Since these databases can grow quite quickly, some sort of system that would cull uninteresting entries (i.e. those that are synced across all devices) would be almost certainly necessary.

Another challenge in visualizing the processes would be how to succinctly represent the messages while they are being passed. Again we want to show as little information as necessary to make it clear what is going on, lest we risk confusing the user with too much information.

Here an intuitive approach would be to simply display the originating router and the sequence number of the LSA currently being sent. This is enough to uniquely identify router LSAs. For the other LSA types, the endpoint of the link would also need to be displayed.

5.3 Optimizing the next hop calculation

Chapter 4 demonstrates quite clearly that the next hop calculation (and in particular the Warshall-Floyd algorithm) is where most of the runtime lies. A more efficient calculation would be very desirable, especially in regards to using the simulation outside of teaching and more for investigating the behavior of a theoretical network.

One approach we could take here is to work with a scheme that allows us to cull some of the problem space. If for example, our cost to get to a certain destination is lower than the cost to get to the link we learned something new about, that change cannot affect our next hop (assuming we only have positive linkweights).

Chapter 6

Summary

Ultimately while we did not succeed in fulfilling all the goals we had set out to begin with, we still consider this a fairly successful undertaking. We have gained deeper insight into the OSPF protocol and managed to model it in a fashion that allows for easier analysis of the protocol.

Furthermore, we have established a foundation for others to build upon and further refine the implementation of OSPF in this simulator.

Bibliography

[1] Knight, S., Nguyen, H., Falkner, N., Bowden, R., and Roughan, M. The internet topology zoo. *Selected Areas in Communications, IEEE Journal on 29*, 9 (october 2011), 1765 –1775.

[2] Moy, J. OSPF Version 2. RFC 2328, Apr. 1998.

[3] Schneider, T., Birkner, R., and Vanbever, L. Snowcap: Synthesizing Network-Wide Configuration Updates. In *ACM SIGCOMM* (online, 2021).