Master thesis
Implementing and simulating the cross-entropy ant system

Jonathan Brugge

July 1, 2010

Contents

Preface

This report describes the work I have done as my graduation project at NTNU in Trondheim. The work was performed within the Telematics department of the university under supervision of Poul Heegaard and Laurent Paquereau. At my home university in Enschede, Pieter-Tjerk de Boer was my first supervisor, with Geert Heijenk taking the role of second supervisor.

I would like to thank them for their help with this project. Pieter-Tjerk always spots the even smallest issues, whether they are mistakes in some formula in the research project or layout problems deeply hidden in the bibliography. I enjoyed the mix of discussing those issues and discussing everything else - from university politics to bicycle holidays. Thanks for that!

In Norway, Poul and Laurent took over - and kept doing that, even when results took longer than expected to arrive and graphs managed to deviate from the expected results in a surprising number of different ways. Extra meetings to find the last bugs and an Easter holiday spent writing a paper with Laurent - my project definitely took more of your time than the standard NTNU thesis. Thanks for all your help along the way!

Jonathan Brugge

June 2010

Introduction and contributions

For this thesis project, I have worked on the Cross-Entropy Ant System (CEAS), a routing protocol developed at NTNU. CEAS had been implemented in network simulator ns-2 [1]. At the start of the project, the plan was to

- Implement CEAS in ns-3
- Verify the implementation against the existing ns-2 implementation
- Come up with improvements to CEAS for use in environments with a relatively high rate of changes in the network topology
- Simulate the improvements to verify them

Ns-3 is a new simulator, written by the developers of ns-2, but otherwise incompatible.

During the project, it became clear that porting to ns-3 was more work than originally expected. Thus, a relatively big part of this report is dedicated to the work done to implement CEAS in ns-3. The last part is about the additional scenarios that have been simulated with the ns-3 implementation and the specific improvements that have been made.

Where applicable, parameters used for the simulation have been included in this report. An earlier study [2] found that many studies in the field of network simulation lack information about the parameters, making it difficult to verify the results. The source code used for all simulations is available on request.

Together with Laurent and Poul, I have written a paper about the experience of porting CEAS from ns-2 to ns-3. The paper has been accepted at a conference and is included in this report as appendix B. Appendix C contains part of the presentation about ns-3. I prepared and presented it as part of a course for PhD students at NTNU, organized by Poul.

Structure

This thesis starts with an introduction to CEAS, including existing extensions to the system. It is followed by an introduction to ns-3 in chapter 3 and 4. Chapter 5 describes how CEAS has been implemented and how it fits in ns-3. The implementation is then validated against the existing ns-2 implementation in chapter 6. Some remarks about ns-3, derived from the experience of implementing CEAS in it, form chapter 7.

Improvements to CEAS to make it more usable in congested networks are discussed in chapter 8, which also includes simulations of such networks. Conclusions follow in chapter 9, after which some appendices are included.

Contributions

The original goal of the project was to improve CEAS by making it more suitable for highly dynamic networks. Others have worked on different projects to improve CEAS: the most important examples are the addition of a system called *elite selection*, described in [3], and the subpath extension, introduced in [4]. Both extensions are described in section 2.2.5 of this report.

Though some work has been done to adapt CEAS for use in such networks, an important part of this thesis project is the experience gained from implementing the protocol in ns-3. Because the protocol had been implemented in ns-2 before, it was possible to compare the two implementations and with that, the two simulators. As many researchers will have to decide whether to use ns-2 or ns-3 for future projects, the comparison can be useful to others. I will present the paper Laurent, Poul and I wrote about that decision at SIMUL 2010, a conference about simulations in August 2010 in Nice.

Part I

The Cross Entropy Ant System

Chapter 1

Background

Requirements to networks are changing over time and with those, the requirements to routing protocols. New developments mean that it is not necessarily enough to find a single shortest route from a source to a destination, but that specific requirements are placed on the quality of service offered by different paths and the possibility to balance the load between paths.

Different approaches are used to develop new routing protocols that fulfill those requirements. One approach is the use of algorithms inspired by the behaviour of swarms of animals, specifically ants [5]. A protocol based on how ants find their way to food was proposed in [6].

One example of an ant-based routing system, designed with the new requirements in mind, is the Cross Entropy Ant System (CEAS). Cross-Entropy Ant System (CEAS) is a fully decentralized Ant Colony Optimization (ACO) system [7] first introduced in [8] and developed at NTNU. Different from some other systems, it uses the cross-entropy method as a mathematical basis.

This thesis does not contain an in-depth description of the cross-entropy method: it has been described in the research project, which was written as a preparation for this thesis project and is attached to this report as appendix D. An extensive introduction to the cross-entropy method can be found in [9].

ACO systems are systems inspired by the foraging behaviour of ants in nature. ACO systems belong to the class of Swarm Intelligence (SI) systems. SI systems are formed by a population of agents, whose behaviour is governed by a small set of simple rules and which, by their collective behaviour, are able to find good solutions to complex problems. ACO systems are characterized by the indirect communication between agents - ants - through local modifications of their environment, referred to as stigmergy and mediated by (artificial) pheromones. The pheromone trails reflect the knowledge acquired by the colony and good solutions emerge as the result of the iterative interactions between ants.

CEAS was originally designed for distributed network path management and one of the successful applications is adaptive routing in dynamic networks [10]. Given different study objectives, the system has been implemented in different programming languages and platforms. CEAS has also been implemented in a testbed, which makes it possible to validate simulation results. It also provides useful insights in the complexity of swarm-based methods in real routers and reveals potential implementation challenges and performance bottlenecks which are hard to predict through simulations and analysis alone.

Chapter 2

The Cross Entropy Ant System

The Cross Entropy Ant System (CEAS) is a routing protocol developed at NTNU that takes ideas from nature, specifically how foraging ants find their way, to build an efficient and robust routing protocol. A basic description can be found in [11]. It is an example of a swarm-based routing protocol. CEAS uses the cross entropy method to quickly converge to good routes.

Note that the following is not a complete description of CEAS: it is intended to provide the information needed to understand the rest of this thesis. More information about CEAS cna be found in [11] and its references.

2.1 Foraging ants

CEAS takes its basic idea from nature. Consider an area with an ant nest and a place with food, respectively called "s" and "d" for "source" and "destination". Each individual ant tries to find a way from the source to the destination. While travelling, it leaves a trail of pheromones. Those pheromones evaporate over time. Thus, if an ant uses a short route, it will pass there more often per unit of time on its way from the source to the destination and back again, resulting in a high concentration of pheromones on that route.

New ants choose their route partly based on the pheromones: the chance of taking a specific route is proportional to the amount of pheromones. The result is that the most efficient routes are used more often, which results in even more pheronomes on that path. Longer routes are chosen less often and eventually, the pheromone concentration gets lower and lower. The effect is clear: something which started as a random walk converges to an efficient path between source and destination. This phenomenon, where a group of systems do

not communicate directly with each other, yet specific behaviour for the whole group emerges, is called emergent behaviour.

2.2 CEAS

2.2.1 Introduction

A number of routing protocols, such as AntHocNet, are designed after the foraging ant behaviour described before. CEAS does that as well, but - contrary to the other protocols - bases it on the cross entropy method, which provides a mathematical basis for CEAS. The basic method is described in [12], though it is not directly applied to CEAS there. An overview of the algorithms used in CEAS can be found in [11].

The basic routing information in the protocol is a value $p_{t,r,s}$, which is the routing probability that a packet will go to node s, when it is at node r at iteration t. For all nodes, these probabilities can be grouped in a matrix $\mathbf{p}_t = p_{t,rs_{\forall rs}}$. The routing probabilities are the 'digital pheromones' of the protocol: the higher the pheromone value from r to s, the higher the probability that a packet will travel over that link as its next hop.

The protocol also uses a *temperature* γ_t, which converges to a value based on the cost of the best (lowest cost) route between two nodes. A performance function $h(\mathbf{p}, \gamma)$ indicates how good a certain matrix $\mathbf{p}$ is.

The algorithm works as follows:

1. Start with a set of routing probabilities $\mathbf{p}_{t=0}$ that are for instance uniformly distributed: all paths have the same probability $1/n$ with $n = size(s)$.

2. Generate a number of sample paths and select those samples that give the best results - so find a γ as low as possible, that still gives a certain number of cases that satisfy $h(\mathbf{p}, \gamma)$.

3. Using those samples, generate a new matrix $\mathbf{p}_t$ which is as similar as possible to the optimal matrix. This optimal matrix is a matrix such that routes generated with it are the shortest routes, i.e. the matrix which, when used, results in the lowest possible temperature γ.

4. Increase t by 1 and repeat the procedure with the new matrix. Stop when the temperature γ stabilizes.

For $t \to \infty$, $\mathbf{p}_t$ gets an optimal solution, where $p_{t\to\infty,r,s}$ is either 1 (if the link between r and s is on the path with minimal cost) or 0 (if it is not). The

performance function used in CEAS is $h(\mathbf{p}, \gamma_t) = e^{-\frac{L_t}{\gamma_t}}$, with L_t the cost at time t. A more in-depth description can be found in section 2.3 in [3].

This procedure needs a number of samples before it can update the temperature and the matrix, which is not practical in a routing environment: it would be better if the probabilities would be updated with each arriving routing packet, instead of waiting for a batch of packets to arrive. CEAS achieves that by adjusting the performance function every time new information arrives, basing it on both the new information and the currently used function.

2.2.2 Implementation

To implement the behaviour described above, the routing protocol uses packets called *ants*. Ants travel from a source to a destination, possible using the matrix $\mathbf{p}_t$ to choose the route, accumulating the cost of the followed path. At the destination, the temperature is updated. The ant then travels back along the same way to the source, updating $\mathbf{p}_t$ in the nodes it passes.

2.2.3 Avoiding converging to a local optimum

With the algorithm described above, there is a risk of ending up with a path that is not actually the shortest. At first, a reasonable - but not optimal - path p_R may be found. No other paths have any pheromone at all, so all ants choose to follow p_R. No better path is ever found.

To avoid that situation, CEAS knows two different kind of ants. *Normal forward ants* use the pheromone tables to find their destination, while *explorer ants* simply pick a random next hop, not using the pheromone values at all. Using a suitable mix of normal ants and explorer ants results in new routes being found, while known routes are reinforced with the right amount of pheromone.

That does not solve the whole problem, however. Consider the situation where the reasonable path p_R is known. At some point, a shorter route p_B is found. However, the chance of taking that route is not very high, as only one (explorer) ant 'deposited' pheromone on that route, compared to possibly thousands of ants on p_R. Thus, p_B is not chosen more often and its pheromone value is not reinforced.

To avoid this situation, a basic solution is to sample a number of random paths at first, before starting to use the pheromone values. Thus, the default implementation of CEAS has an *initialization phase* during which only explorer ants are sent. The pheromone tables then contain reasonable levels, after which the normal system with both normal and explorer ants can take over to maintain the short routes and adapt to changes in the system.

2.2.4 Cycles in paths

Given how ants find their way to a destination, it is possible that an ant visits a single node twice on its way to the destination. In that case, the path to the destination will contain a cycle. In most cases, such cycles are not a problem as the system will converge to a route without cycles: a route with cycles can not be the best route to a destination, assuming a positive cost for every hop in the path.

In some cases, cycles can be a problem. Updating paths which contain a cycle means that ants have to travel more, resulting in a higher network load. Also, investing 'energy' in longer paths might cause a longer time to converge to the shortest path. For some possible extensions to CEAS, such as subpaths (discussed in 2.2.5), the effects can be noticable. In [13], three different approaches to handling cycles in CEAS are discussed. For the simulations in this report, cycles are detected at every hop and packets are dropped if they contain a cycle.

To avoid the creation of cycles, an extra mechanism has been used. Before the next hop of an ant is chosen, all nodes that it has visited before are taken out of the list of possible options. Though this heuristic has limitations in the current ns-3 implementation of CEAS, it prevents the most common cases, such as ants travelling back to the node from which they just came.

2.2.5 Extensions

The system described above works, but it is not as efficient as it could be. A number of improvements are already known that speed up convergence or reduce the overhead of the protocol by limiting the number of ant transmissions needed to get good routes.

Elite selection is one such optimization. When a forward ant reaches its destination, it is only converted to a backward ant if the cost for the path it has followed is within a certain distance from the best path known to that destination. Other ants are discarded. In this way, ants that would only confirm that a low-chance path is indeed not worth taking are not propagated, thus causing a reduction in overhead. Elite selection also helps to avoid the problem of converging to a local optimum, described before. A more thorough description of elite selection, including benchmarks, can be found in [3].

Another optimization can be found in the rate ants are generated. When a network is changing, a high number of ants is needed to find the best (new) routes. However, if no changes occur, it would be better to create fewer ants. To solve this problem, ant rates are self-tuning. If almost no forward ants return to the source node, that node knows that the protocol has likely not converged to a stable state yet - ants either do not get to the destination at all or if they get there, they are not converted to backward ants because they do not qualify as elite ants. In a stable network, most ants reach the destination over the

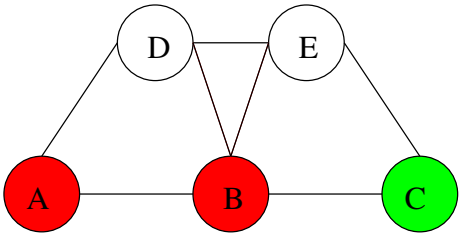


Figure 2.1: Five node network. The red nodes A and B are the source of some network traffic. The green node C is the destination of traffic from both source nodes.

lowest-cost path and are thus within the elite selection range. That implies that the number of ants that is generated can be lowered when the number of forward and backward ants is almost equal.

A third extension to CEAS is the introduction of *subpaths*, introduced in [4]. The idea is to limit the number of ants by sharing information between different nodes. In basic CEAS, routes are found for a source-destination pair. In figure 2.1, both node A and node B need a route to destination node C. Clearly, the shortest path from A to C leads through B. It is also clear that at node B there is knowledge about the shortest path to C. In basic CEAS, the information discovered by ants from B would not be used by ants travelling from A. The subpath extension makes it possible to share that information, resulting in less overhead. This is done by cutting the path from the source to the destination in smaller pieces. For each piece, the pheromone values are updated individually.

Part II

Porting CEAS to ns-3

Chapter 3

Introduction

A big part of the time available for this thesis has been spent on porting the CEAS protocol from ns-2 to ns-3. The following sections provide information about what has been done exactly and the issues encountered along the way. Ns-3 is described in chapter 4, followed by an overview of the design chosen for CEAS in ns-3 in chapter 5. Results obtained from the ns-3 implementation are then validated against the original ns-2 implementation in chapter 6. Finally, chapter 7 discusses the experience of using and extending ns-3.

The design description in chapter 5 is necessarily somewhat abstract. Readers interested in implementation details can take a look at the code, which is extensively documented and available on request.

Chapter 4

About ns-3

This chapter describes the properties of the ns-3 simulator, as presented by the ns-3 developers. In chapter 7, the actual experience with the simulator is discussed.

4.1 Introduction

Ns-3 is a network simulator, developed by mostly the same group of people that work on or have worked on ns-2. It got started because there are problems with the ns-2 design that they felt could not be fixed without breaking compatibility with earlier versions [14].

Specifically, those problems included:

Bi-language system (C++/tcl) Ns-2 uses both C++ and tcl to build simulations. The combination of both languages is difficult to debug and a barrier for new developers.

Scalability problems Tests show that ns-2 does not scale well to large simulations, making it unsuitable for some research.

Core packet structure In ns-2, packets are not serialized and deserialized. To be able to run simulations against real-world systems, support for 'real' packets is needed.

Lack of validation and verification Many models in ns-2 are not validated against the real world, which causes users to doubt whether simulation results are the same as the results that would be measured in real-world implementations.

To fix these problems, the ns-3 project was started in 2006. The first stable version was released in June 2008. The current release, as of May 2010, is ns-3.8. New versions are released every three to four months.

4.2 Features

To avoid the problems that developed in ns-2, a number of specific features have been introduced in ns-3. Those are described in the next sections.

Extensible core

Ns-3 is written in C++. It features an optional Python interface. In contrast to ns-2, users don't necessarily have to know both languages. To make it easy to extend the simulator, its goal is to be documented more consistently than ns-2. Also, the coupling between different models has been minimized by using object aggregation.

Attention to realism

In ns-3, nodes in the simulation resemble real computers more than in ns-2. Every node is constructed out of devices and there is an internet protocol stack that closely resembles the stack on real systems. Applications written in ns-3 can use an implementation of the BSD socket API.

Software integration

Compared to ns-2, it has become easier to interact with other software packages. Network traffic generated by ns-3 can be traced and written to a file in the pcap format, which makes it possible to analyze it with tools like Wireshark.

Support for virtualization and testbeds

Because of its attention to realism, ns-3 can be used as a virtual system or in testbeds. Real applications can run on top of a protocol stack implemented by ns-3. It is also possible to run ns-3 applications on real networking stacks or to run an instance of ns-3 in a network, where it interacts with 'real' systems. Such features make ns-3 more of an emulator than a traditional simulator.

Tracing and statistics

In ns-3, a tracing framework makes it possible to decouple tracing as much as possible from the simulation code itself. The code can provide tracing hooks, which can then be connected to tracing sinks by the user. A tracing hook can be the arrival of a packet, the occurence of a certain error state or anything else. Tracing sinks can analyze the incoming data, aggregate it and store it in different formats, depending on the needs of the user.

A statistics module is also included in ns-3, though its current feature set is rather limited. As part of this thesis, the functionality has been extended.

Architecture

Ns-3 consists of a core simulator part and a number of layers that add the networking-specific elements. It provides an internet stack with implementations of protocols like TCP and UDP, as well as lower-level protocols like various versions of 802.11. Different components and applications can be added to nodes, after which nodes can be connected to each other. To help with building up nodes and creating a network topology, helper scripts are provided. A schematic view of ns-3 is shown in figure 4.1.

Compared to ns-2, there are other differences as well, such as the build system. Section 7.1.2 discusses the differences in more detail.

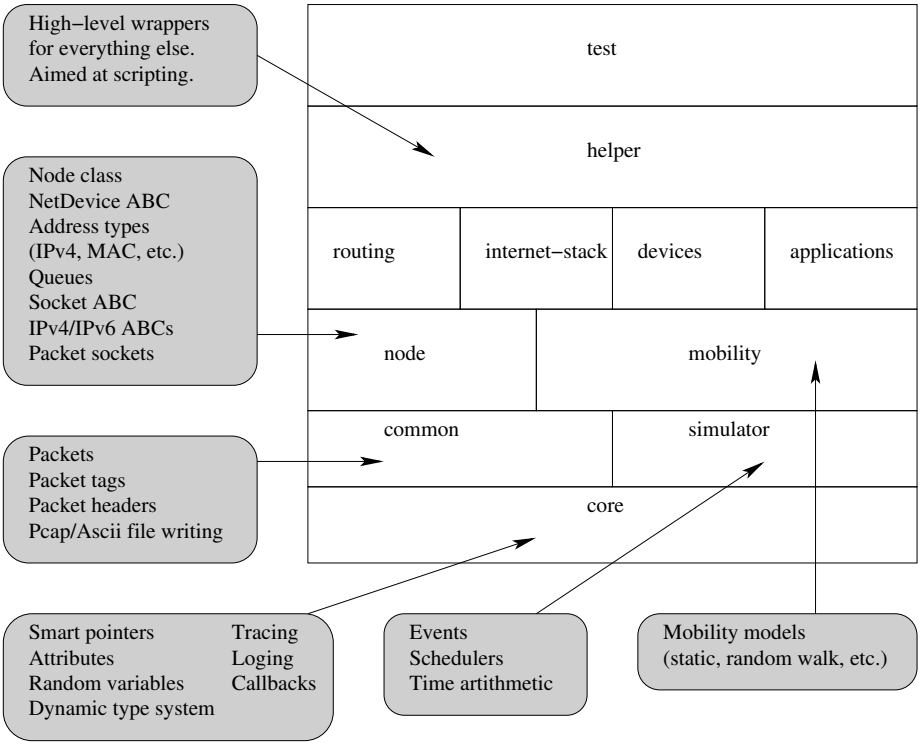


Figure 4.1: Schematic view of ns-3 architecture, based on figure from [15].

Chapter 5

CEAS in ns-3 - the architecture

Soon after the thesis project started, it was decided that porting CEAS to ns-3 would be the first step. It was expected that ns-3, being a more modern project and developed by people that knew the advantages and disadvantages of ns-2, would make it easy to extend and improve CEAS. Thus, the time spent on implementing CEAS could be won back by being able to develop improvements in a faster way. In the end, that was not what happened: porting to ns-3 took longer than expected and the extra time could not be compensated by faster development later on.

The CEAS code in ns-3 uses the same algorithms as those used in ns-2. Most of it has, however, been built from scratch in the ns-3 framework. The parts related to the temperature calculations could be taken from ns-2. Some inspiration for the design came from the implementation of the Optimized Link State Routing (OLSR) protocol, which is the only other decentralized algorithm currently implemented in ns-3. The first part of this chapter describes how the ns-3 version of CEAS is designed. The last section is dedicated to the infrastructure used to perform simulations with ns-3 and is not specific to CEAS.

5.1 Overview

The implementation consists of a number of building blocks. Most important is the class `CeasRoutingProtocol`, which implements the protocol. To be able to work, each instance of the protocol maintains a pheromone table. This table works as a routing table and is used to perform the actual routing. Also, every node which is a destination of some traffic keeps track of temperatures with a temperature table. Ants are modelled in separate classes: every ant is a separate packet. Those classes are responsible for the creation, serialization and

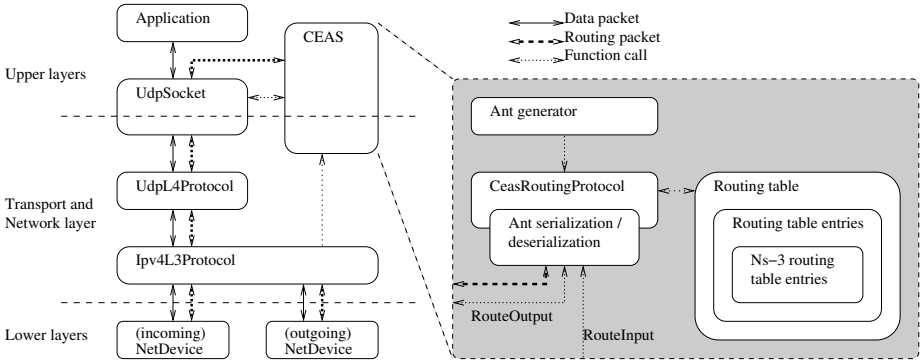


Figure 5.1: Design of CEAS within the ns-3 framework.

deserialization of ant packets. Figure 5.1 shows how CEAS has been integrated in the ns-3 framework.

5.2 Temperature table

The temperature table, modelled by a class `CeasTemperatureTable`, provides methods to add temperature entries to a table and retrieve them. Entries are instances of `CeasTemperatureTableEntry` and have a key, which is by default a source-destination pair, that identify entries in the table. The original default in ns-2 is to use an identifier that specifies the pheromone type. Other key types can easily be added in both the ns-2 and ns-3 implementation.

The `CeasTemperatureTableEntry` itself contains both the normal and the elite temperature γ of a key and the 'search focus'ρ. The temperatures, instances of `CeasTemperature`s, consist of the actual temperature γ and the parameters a and b, which are used to update γ.

5.3 Pheromone table

Ns-3 provides a class `Ipv4Route`, which models IPv4 routing table entries, and methods to create and update those entries. However, the fields in such an entry are not sufficient for CEAS. For that reason, a new class `CeasRoutingTable-Entry` has been added. The class uses ns-3's IPv4 routing table entries where possible.

The resulting structure is shown in figure 5.2. The outermost entity is a table with entries. The key for every entry is an IPv4 address, which is the destination of that entry. The table provides methods to create, read, update and delete routing entries.

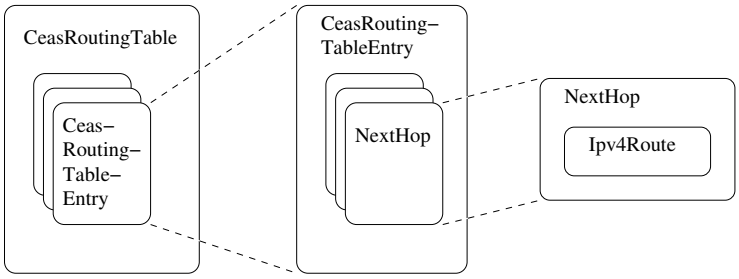


Figure 5.2: Routing table elements of the CEAS implementation.

Every entry has a set of `NextHop`s, which are possible next hops to reach the destination. It is possible to ask for a `NextHop` instance in two ways:

- Using stochastic routing with the pheromone values as input, as used by normal forward ants.
- Using a random selection of the next hop, where all hops have equal chances, as used by explorer ants.

`NextHop` is modelled as a wrapper around ns-3's routing entries. The wrapped entries are used to store the IPv4 address of the next hop and the outgoing interface address to reach that address. The values that can not be stored in the `Ipv4Route` class provided by ns-3, such as the cost for this link and the pheromone value associated to it, are stored in the `NextHop` itself.

5.4 Ant packets

In ns-2, contrary to ns-3, packets are not serialized. For that reason, there was no clear definition of the packet format to use for ants in ns-3. An earlier implementation of CEAS in AntPing [16] used the IPv4 route record mechanism, described in [17] to store route information. Disadvantages of that approach are the limited number of hops that can be stored (no more than nine) and the limited amount of information that can be stored (just the IPv4 address). For the ns-3 implementation, a new packet format has been designed that does not have those limitations.

The implementation consists of a header which is common to all ants and specialized headers for neighbour discovery packets, forward ants and backward ants. The neighbour discovery packets will be discussed in section 5.4.1. Forward ants are again split in normal forward ants and explorer ants. The common header simply contains a value which specifies the type of ant. When serialized, any ant thus consists of two headers: the 'outer' header, which just specifies the ant type, and the 'inner' header, which - for ants - specifies all the other

Field	Size	Value
Source address	4	The IPv4 address of the source.
Destination address	4	The IPv4 address of the destination.
Cost	8	The cost to get from source to destination.

Table 5.1: Packet structure - hops. Sizes are in bytes.

information that has to be encoded in an ant. This structure has been chosen because it makes it much easier to deserialize ants.

The structure that has been used for normal forward ants is shown in figure 5.3.

Each 'inner' ant header contains the following fields:

Size (2 bytes) Not used so far.

Ant type (1 byte) Not used so far. This could be used to have different 'types' of routes between two nodes, for instance one route which has a low latency and another route which is optimized to have a maximal bandwidth. In that case, the source and destination address are not enough anymore to keep the different ants apart. Note that this field does *not* specify whether the ant is a forward ant or a backward ant - that is encoded in the common, 'outer' header.

TTL (1 byte) The time-to-live (in number of hops) of the packet.

Destination (4 bytes) The IPv4 address of the destination of the packet.

Gamma (8 bytes[1]) The gamma value of the packet. This is the temperature γ, which is used to update the pheromone values. It is only relevant when the ant is travelling back to its source.

L (8 bytes) The cost L of the ant. In the current implementation, it is only relevant when the ant is travelling back to its source. Future extensions could use it to drop a forward ant based on the cost it has accumulated so far.

Rho (8 bytes) The search focus ρ of the packet. This value is used to update the temperature γ at the destination.

Number of hops (1 byte) The number of hops the packet has already taken.

Following the 'number of hops' field, descriptions of the individual hops are added to the packet. The structure of the description is shown in table 5.1.

The packet structure could be optimized. For instance, there is no reason to use eight bytes per floating point field in the packet: two bytes would probably provide enough precision. However, to keep extensions to and debugging of the protocol easy, optimization of the packet size has not been an objective.

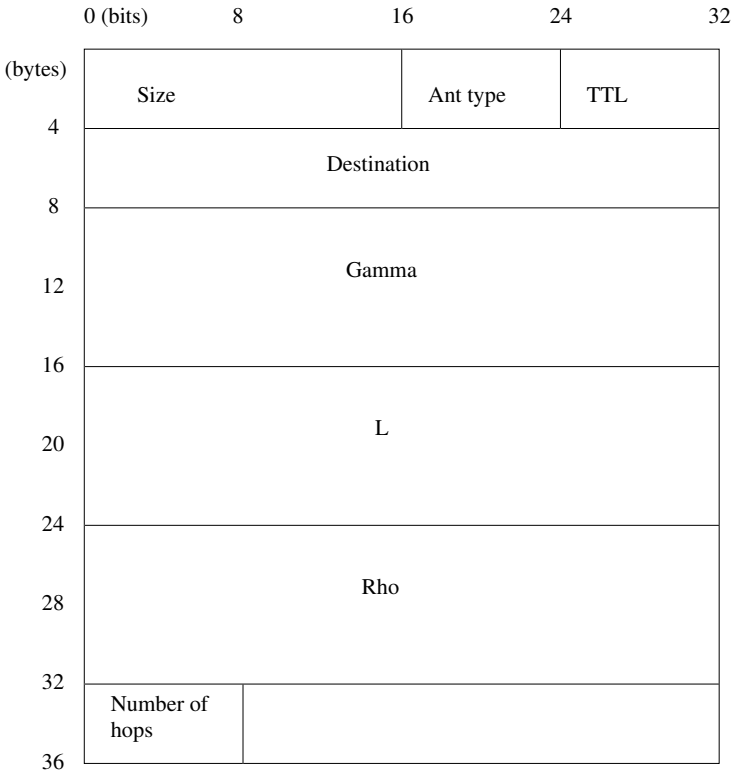


Figure 5.3: CEAS packet format in ns-3.

The encoding of the different hops the packet has passed deserves some attention. In the current CEAS implementation, both the source and the destination address for each hop are stored in the packet. It is possible to store just the source addresses. However, it is not enough to store just the destination addresses. The reason that the source addresses have to be stored comes from how backward ants find their way. If a forward ant travels from node A via node B to node C, a backward ant will be generated at C. This backward ant needs the IP address of the *outgoing* interface from B to C: that address is used to determine which outgoing interface from C to B has to be used.

Future extensions to the protocol might need both the source and destination addresses of each hop, for instance when the connections between nodes are not point-to-point links. For that reason and for easier debugging, both the source and destination address have been included in the ants.

For basic CEAS, it is not needed to include the cost of each link: just incrementing L at every node on the way would be enough. However, to implement the subpath extension to CEAS, the cost of every hop has to be known. Other extensions might also need the cost of individual hops, so it has been included in the packet structure. However, just like with the other floating point fields, it could be made significantly smaller than it currently is.

Methods have been implemented to change values in a packet, serialize it and deserialize it.

5.4.1 Neighbour discovery packets

In the ns-2 implementation of CEAS, nodes know which neighbours they have. In real networks, that is not necessarily true: neighbours have to be discovered somehow. The ns-3 implementation of CEAS contains an extra packet type, called 'hello' packets, to discover neighbours. Packets are periodically sent through all connected interfaces and when a node receives such a packet, it sends a reply. When a node receives a reply, it adds the sender of the reply packet to the list of known neighbours. If a known neigbour does not reply to a configurable number of discovery packets, it is removed from the list of known neighbours.

The ns-2 implementation does not contain such a mechanism. To make it possible to compare the two implementations, the discovery mechanism has been disabled and nodes are configured with a list of known neighbours at the start of each simulation. For more realistic simulations, the neighbour discovery packets can be enabled easily.

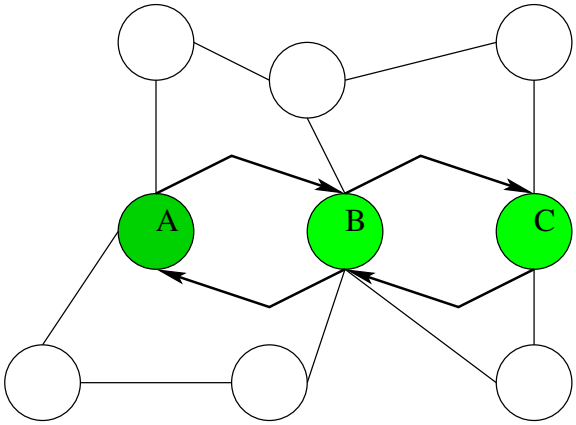


Figure 5.4: Network with nodes A, B and C.

5.5 Routing protocol

Assume a network as shown in figure 5.4. This section describes how the routing protocol is implemented by following a single ant on its way from node A via node B to node C and back through B to A again.

5.5.1 Before sending the ants

Before any ants are sent, a basic infrastructure has to be set up. A list of neighbours is needed, as well as a mapping from the IPv4 addresses of the neighbours to the corresponding UDP sockets used to send routing information.

Also, an - initially empty - list of 'needed' destinations has to be created. This list is used to determine which routes have to be discovered. Using such a list allows for both a reactive and a proactive approach to routing. In the first case, destinations are added when a data packet requests a route to that destination. In the second case, external code can add destinations to the list, which will then automatically be looked for by ants.

5.5.2 Leaving node A

Ants are generated at regular intervals with a configurable rate. There are separate rates for normal ants and explorer ants. A new ant picks a next hop from the pheromone table. For normal ants, the pheromone values are used to influence the chance of choosing a certain neighbour as the next hop. For explorer ants, every neighbour has an equal chance.

Depending on the chosen next hop, a certain cost is associated to the ant. The

various other fields, such as the time-to-live of the ant, are set to the right values. The packet is then queued for transmission.

When the transmission timer expires, all queued packets are sent. In the current implementation, the transmission timer is set to zero every time a packet is queued. The result is that all packets are sent individually. The timer could be used to send bursts of packets, which might be useful in for instance wireless scenarios.

For every packet that is to be sent, a timestamp is added. This is a tag that is just used for simulation purposes: it is not serialized with the normal packet and thus does not add to the transmission delay. After completing the packet, the socket associated with the link to the next hop is looked up. The packet is then sent through the socket to be processed by the lower layers in the communication stack.

5.5.3 Passing through node B

When the ant arrives at node B, the first part of the header is read. This part contains the identification for the different ant types. Depending on the type of ant, different handling routines are called. In this case, the handler will conclude that the ant has not arrived at its destination yet and that it has to forward it.

Next, some basic checks on the route the ant has taken are performed. The time-to-live should still be above zero and the ant should not have visited the node before. In both cases, the packet is dropped. Then, a next hop for the ant is chosen using the process described before. One difference is that all nodes which have been visited before by the ant are blacklisted: they can not be chosen as a next hop. Note that this does not completely prevent loops: neighbours may have multiple IP addresses, not all of which may be known at B. Thus, the loop check is still needed. Normal forward ant packets are dropped if no valid next hops are found. For explorer ants, that is different: if all neighbours are blacklisted, the blacklist is ignored and the explorer ant is sent to one of the neighbours, where it might find a node which it has not visited yet. The different way to handle explorer ants prevents dropping such ants too often, which would make it difficult to find routes when the discovery process has just started. The ns-2 implementation does not limit this mechanism to explorer ants and handles normal forward ants in the same way.

The destination of the packet is then added to the local list of known destinations, which will be needed when the ant returns, and the cost and path information within the ant are updated. The ant is then serialized and sent as described before.

5.5.4 Arriving at node C

At C, the packet is analyzed in the same way that happened at B. This time, the system concludes that the ant has arrived at its destination. The temperature value γ is updated for the specific source-destination pair, in this case the route $A \Rightarrow C$. If the route is within certain limits of the best known route, i.e. it is an elite ant (see section 2.2.5), a backward ant is generated.

Backward ants are basically identical to forward ants, except for a field in the header indicating that the ant is on its way back. Thus, the forward ant is simply copied into the new ant and only the first part of the ant is newly generated. The packet is then queued to return to A.

5.5.5 The way back

The ant arrives at node B again, where the pheromone values for all routes to C are updated, based on the values of γ, L (the cost) and β. The packet is then forwarded to A, based on the route stored in the ant.

At A, the same happens, except for the forwarding: it has arrived at its destination and is dropped.

5.6 Infrastructure

To perform the actual simulation, a number of helper tools have been written. These tools are not specific to CEAS. A simulation run consists of the following steps:

- A simulation script (written in bash/sh) is started.
- The script compiles and runs the simulation scenario file (written in C++) a configurable number of times with the specified options.
- The simulation scenario stores the results in an SQLite database.
- The simulation script runs queries on the SQLite database to get data points.
- The simulation script calls gnuplot to convert the data points to graphs.

Scenario files contain the definition of the network topology and specify any special events, such as link losses, that occur.

During development, it became clear that the SQLite support of ns-3 had performance problems. The problem could be tracked down to the lack of transaction

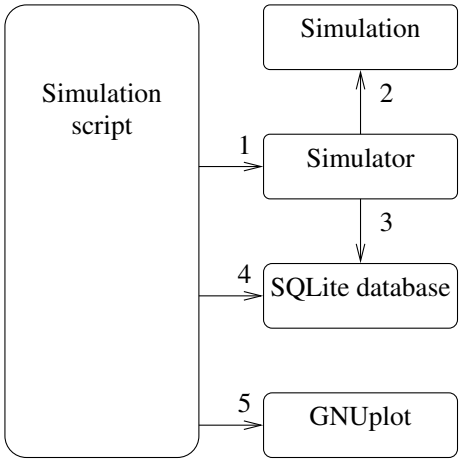


Figure 5.5: Workflow of simulating CEAS with ns-3.

1. The simulation script starts the simulator with specific simulation parameters.
2. The simulator runs the simulation.
3. The simulator stores the results in a SQLite database after every simulation run.
4. The simulation script queries the database to get the results.
5. The results are plotted with GNUplot.

support in the ns-3 interface to SQLite. The problem has been fixed and the resulting patch has been provided to the ns-3 developers[2]. It is integrated in new ns-3 releases.

To be able to measure the changes in temperature and pheromone values over time, the statistics framework of ns-3 has been extended. Some scripts are used to run a simulation, store the results in a database and extract relevant data from the database to generate graphs. The process is shown in figure 5.5.

[2]See http://mailman.isi.edu/pipermail/ns-developers/2009-November/006959.html and replies.

Chapter 6

Validation

To assess whether the CEAS port to ns-3 behaves identically to the latest ns-2 version, a number of simulations have been run on both simulators. The results show that the ns-3 code behaves exactly like the original version.

To validate the convergence and temperature calculation, a network consisting of 12 nodes has been chosen. A very similar network has been used in [11] and for the simulation of the subpath extension to CEAS [4]. Because of the experience with the network layout and the available results from ns-2, it was chosen as the network for validating the ns-3 CEAS implementation. In the network, traffic flows from node 10 to node 11. The path with minimal cost is along $10 - 0 - 4 - 6 - 9 - 11$, which has cost $0.0038s$. Figure 6.1 shows the network layout. There is one difference between the network used for ns-3 and the network used before: nodes 10 and 11 have been added. This has been done to have a single interface on both the source and destination node, which makes it easier to run the simulations with the ns-3 implementation. The cost of the links connecting the extra nodes to the original network have been set to zero, so the cost of the path is the same in both networks.

The expected result is that most ants converge to this route. The performance function used in CEAS is $h(\mathbf{p}, \gamma_t) = e^{-\frac{L_t}{\gamma_t}} = \rho$ (see section 2.2.1). In this function, L_t the cost at time t, γ_t is the temperature at that time and ρ is the 'search focus' of the protocol, which is 0.01 in both ns-2 and ns-3. With some rearranging, a theoretical lower bound for the temperature can be found, given values for ρ and L (dropping the subscript t for the variables, as this is about the converged state, not an intermediate time):

$$e^{-\frac{L}{\gamma}} = \rho$$

$$\gamma = -\frac{L}{\ln(\rho)}$$

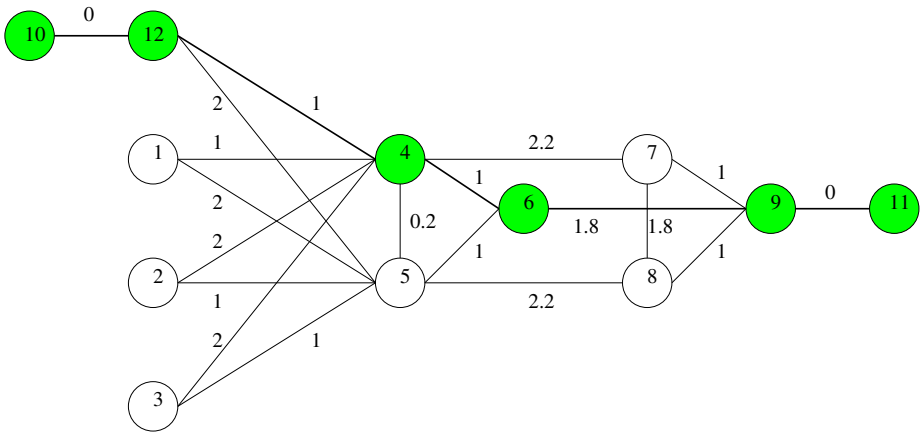


Figure 6.1: The 12-node network used for validation. Costs are delays in milliseconds. The shortest path is indicated.

The total temperature γ at node 11 for the source-destination pair $(10 - 11)$ should gradually converge to $-\frac{L}{\ln(\rho)}$. In both simulations, $\rho = 0.01$ has been used, which gives an expected lower bound for the temperature:

$$\gamma = -\frac{L}{\ln(\rho)} = -\frac{0.0038}{\ln(0.01)} \approx 0.000825$$

Due to explorer ants taking different routes, the temperature of the system will stay slightly higher than 0.000825. The elite temperature, which is only updated by ants with a low enough cost, should always be lower than or equal to the total temperature, but has the same lower bound.

Figures 6.2 and 6.3 show the evolution of temperature over time in ns-2 and ns-3, respectively. The parameters used in the simulation are described in appendix A. The code for updating the temperature has been taken from the ns-2 implementation, so any difference in temperatures between the two implementations comes from other parts of the code.

Figure 6.3 also shows that the elite temperature follows the expected curve: it is always lower than the total temperature, while never getting below the lower bound. The curves from the ns-3 simulation are within the confidence intervals of the curves generated by ns-2.

It was checked manually that the chosen path is indeed $10 - 0 - 4 - 6 - 9 - 11$: the pheromone values at all intermediate nodes converge to the expected values.

A different network topology, consisting of only seven nodes, has also been used to test CEAS in ns-3. It has not been compared to ns-2, but it converged to the expected route and temperature. Based on these results, the ns-3 implementation is believed to behave in the same way as the original ns-2 implementation.

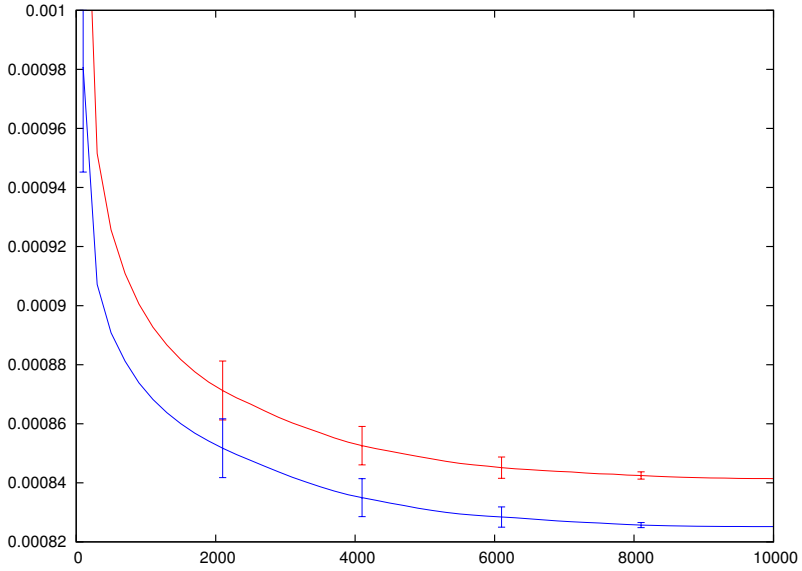


Figure 6.2: Temperature convergence in ns-2, averaged over 30 trials. The red (upper) line is the normal temperature; the elite temperature is shown in blue. The vertical bars are 95% confidence intervals. The figure has been contributed by Laurent Paquereau.

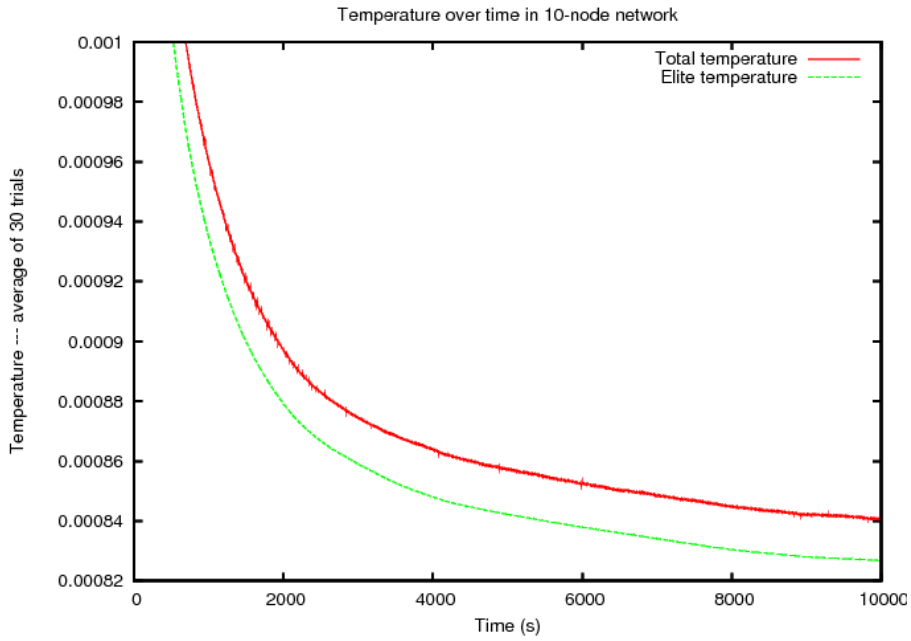


Figure 6.3: Temperature convergence in ns-3.

Chapter 7

Experience with ns-3

At the beginning of the thesis project, I proposed to use ns-3 instead of ns-2 to run the CEAS simulations. The idea was that it would take more time at first, because of the porting of the protocol to a new simulation, but that it would be possible to make up for that by being able to use a more flexible architecture for all simulations.

In practice, that has not exactly happened. The ns-3 architecture is indeed more flexible and would in theory make it easier to run a large number of different simulations. However, the flexibility is mostly concentrated at the application layer. It is very easy to construct nodes, generate all kinds of network topologies and add applications to nodes. Tracing works well too, though it is slightly less useful because of the lack of a complete statistics module: it is possible to get a lot of information out of the system and it is easy to add hooks to get even more data. However, ns-3 does not offer all the needed components to analyze the data easily. For instance, it is possible to run a simulation with certain input parameters and record the (final) value of whatever is to be measured. However, there is no infrastructure to record how that value changes over time within a simulation.

The most important reason for the delay in porting CEAS to ns-3 is the added realism in the new simulator. In ns-2, nodes can have an ID and packets can simply be sent to a specific node. In ns-3, such a basic operation requires a lot more work:

- The nodes have to run an IP stack, with different IP addresses for every network interface. Thus, there is not a single 'node ID'.
- To send a packet, a TCP or UDP connection has to be established by opening a socket to the destination node. The destination has to have the port open: if not, ICMP error messages will be generated.
- Packets are really serialized and deserialized, so it is not enough to simply

pass a structure around in the simulator. Instead, a definition of the packet layout is needed and serialization and deserialization functions have to be written.

The purpose of this research is to improve CEAS, not to get to know the details of the BSD socket API. For that reason, the added realism seems unnecessary - after all, the simulation results show that the ns-2 and ns-3 implementations behave in exactly the same way.

Having said that, there are advantages to the added realism as well. Until now, CEAS did not have a well-defined packet structure. Also, it depended on the existence of a unique identifier for every node, which is not necessarily available in real networks. By forcing realism on the developer, the protocol improves and gets ready for real-world usage. However, for first implementations of a protocol, where the general behaviour is more important than the exact implementation details, ns-3 does not seem the right tool for now.

The experience with implementing a protocol in ns-3 and how it differs from ns-2 has been described in a paper, which is attached to this report as appendix B.

7.1 Comparison with ns-2

7.1.1 Abstraction level

Both ns-2 and ns-3 are packet-based discrete-event simulators, but have different levels of abstraction. Ns-3 mirrors real network components, protocols and APIs more closely. This becomes obvious when implementing CEAS. At the transport and network layers, ns-3 does not abstract any detail. IP and UDP protocols are implemented in detail. A packet is represented as a buffer of bytes and the actual content of a packet needs to be serialized and deserialized. Packets are not simply sent: a socket has to be created and connected and errors have to be handled properly. Trying to connect to a closed port results in an ICMP error message. In ns-2, there is no detailed implementation of either UDP or IP.

The main reason and advantage of this increased realism is to facilitate code re-use, portability and validation. In particular, it makes it to possible to embed the simulator in a mixed environment with real hardware, software and networks. Any problem that would occur when implementing and running a protocol in a real-world system is likely to occur during the implementation and simulation in ns-3. The downside is the added complexity. Ns-3 confronts the developer with low-level implementation details such as socket communication, packet serialization and addressing at a very early stage. If one wants to try out novel concepts, such a level of details may be overwhelming. The higher level of abstraction in ns-2 allows for a quick implementation and testing of new ideas, and is an important reason for its popularity.

As a result, the implementation of the same protocol in ns-2 and ns-3 is significantly different and porting a protocol from ns-2 to ns-3 is not straightforward.

7.1.2 Usability and adaptability

Elements of usability and adaptability are, among others, how easy it is to learn the tool, to extend existing models and to add new ones. This involves many aspects:

Programming language and debugging

Ns-2 is implemented in C++ and OTcl. Each language taken separately is not difficult to use. The difficulty comes from the combination of the two and the concept of split-object. When developing a new protocol such as CEAS, one has not only to implement objects in both C++ and OTcl, but also the interactions between those objects. This task is made difficult by the lack of documentation and debugging tools for the interface between C++ and OTcl. Ns-3 is written in C++ only and, hence, much easier to debug. It offers Python bindings as well, which are identical to the C++ interface. It is not required to use Python and so far, it looks like most users choose to work with C++.

Building

Unlike ns-2, which uses the traditional GNU build system (autoconf, automake, make), ns-3 uses waf [18]. Waf is a much more recent framework, written in Python, that one needs to learn and adapt to when using ns-3.

Documentation

In addition to the ns-2 documentation [19], many tutorials and reports are available, e.g. [20]. However, not all modules are equally well documented and the documentation is in part outdated. Ns-3 is a much younger project and the amount of available resources is consequently much smaller. The development team strives to write and maintain a manual and tutorial as new models are integrated. Nevertheless, the coverage of the documentation is not complete yet, and parts have to be updated according to API changes. During this thesis project, the ns-3 manual more than doubled in size.

Existing code

Ns-2 and ns-3 are open-source projects, distributed under the General Public Licence (GPL). Hence, a way to learn is to read and study existing code. Many models have been contributed to ns-2, but ns-2 code is generally hard to read because: (i) it includes old code for backward compatibility, (ii) many contributions use different coding styles and design approaches and constitute a patchwork of often incompatible models, and (iii) the code is generally poorly commented. In comparison, ns-3 enforces a coding style and a stricter review process before inclusion, which results in more coherent code which is better commented and easier to read. On the other hand, the number of examples is still limited. When the implementation of CEAS in ns-3 was started, the only example of dynamic detailed routing protocol was OLSR.

Modularity

In ns-2, it is not always easy to simply replace a model by another. In the case of routing protocols, models vary depending on the type and numbers of interfaces; see [21]. In ns-3, layers are clearly separated and interfaces well-defined. Replacing objects by similar ones, e.g. a routing protocol, is therefore much easier. On the other hand, the architecture of ns-3 closely maps that of existing systems and implementing untraditional approaches or different levels of abstraction, e.g. abstracting the IP layer, is much more demanding. At the application level, ns-3 is much more flexible than at the transport and network layer.

7.1.3 Built-in components and models

Ns-3's focus on extensibility makes it easier to add new components and properties to nodes, which can be an advantage. The number of protocols available for ns-3 is still relatively small compared to ns-2. However, new protocols are added regularly, particularly in the field of wireless communications. The increased focus on validation makes results obtained with ns-3 potentially more thrustworthy than results from ns-2.

7.1.4 Experiment setup, control and analysis

Both ns-2 and ns-3 provide basic network elements such as nodes and links and make it easy to setup simulation scenarios. In particular, ns-3 provides various helpers to facilitate the creation, initialization and connection of the different entities in the simulation. However, both simulators natively offer very limited support for data collection and experiment control. For instance, neither of them provide mechanisms for transient period detection, specification of termination conditions other than time, handling of replications, or parallel

and distributed execution. For ns-2, several frameworks have been developed and provide some of these features. For ns-3, most of these features are being developed or planned, but not yet integrated.

Furthermore, compared with ns-2, ns-3 provides a powerful framework for tracing internal variables, but, for the time being, misses generic trace sinks. Other advantages of ns-3 include the use of standard formats, such as pcap for packet tracing, and the integration of interfaces to external software such as SQLite. An effort to improve the data collection framework in ns-3 has been announced in March 2010 [22].

Finally, the support for visualization in ns-2 and ns-3 is limited (Nam and NetAnim, respectively), in particular for wireless networks.

7.1.5 Development status

Ns-2 is funded through the ns-3 project, but the core development team is only working on ns-3. Ns-2 only receives maintenance updates and less and less models are contributed. Ns-3, on the other hand, is under active development. It has been available for developers and early adopters since 2008. Most of the generic building blocks are in place. However, not all the core APIs are completely stable yet, which may keep some developers from moving to ns-3. For example, the routing API has undergone significant changes until version 3.6 (October 2009). One should also expect some rough edges. For instance, during the development of CEAS, it became clear to us that the SQLite output interface was a performance bottleneck and had to be fixed by introducing support for SQL transactions. The resulting patch has since then been integrated.

The original NSF project for ns-3 is ending this year (2010) and a lot has already been achieved. However, referring to the project goals [23], there is still much to do, including porting models from ns-2, providing support for data collection, experiment control and statistic generation, extending the visualization support, and integrating ns-3 with external tools such as Click. Recently, a new NSF grant has been announced for the development of "frameworks for ns-3" [22]. The framework will focus on better support for controlling the execution of simulations and analyzing the results. Finally, part of the development of ns-3 is also founded through the Google Summer of Code (GSoC) [24] program.

7.1.6 Efficiency

Though no in-depth benchmarks between ns-2 and ns-3 have been performed as part of this thesis, some comparisons with Laurent Paquereau's work have been done. He has written the current ns-2 implementation of CEAS. We have compared our implementations using the same network as was used for the validation of the ns-3 implementation. In our benchmarks, ns-3 used almost ten times more processing time. Profiling showed that much of the extra time is

Simulator	Run time	Number of events
ns-2	0.94s (0.04)	157511 (6689)
ns-3	8.92s (0.33)	412104 (9098)

Table 7.1: Performance comparison

spent in the 'added' layers: those parts that are not included in ns-2, like socket handling and serialization. The detailed implementation of the transport and network layer require processing time, which is not used in ns-2. To compare memory usage, we created 10.000 nodes with an IP stack and the CEAS implementation. Ns-3 used about half as much memory as ns-2. The difference comes from the fact that the ns-3 node is simply a container and only the required components are instantiated while the ns-2 node is a much more static construct including components that may not be used.

The results are summarized in table 7.1. The run-time and the number of events are averaged over 30 replications. The standard deviation is given in parentheses.

Part III

Simulation and optimization

Chapter 8

Simulation

8.1 Introduction

After the implementation and validation of the routing protocol, a scenario has been defined. Based on the scenario, a number of simulations have been run, using different configurations of the protocol.

8.2 The scenario

To have meaningful simulations, a scenario has been defined. Simulations could then be run with the scenario as a basis and any changes could be compared to the basic protocol more easily.

The defined scenario is a city that provides wireless access to its inhabitants. One example of such a city is Trondheim, where wireless access is provided by Trådløse Trondheim [25].

The city has an infrastructure with different base stations and a (potentially large) group of mobile users. The base stations are connected to each other and route the traffic to 'the internet'. Depending on how the users move, the traffic pattern within the network of base stations changes, which might cause congestion in part of the network.

8.2.1 The layout

For most simulations, the base stations are arranged in a grid. Such a layout fits quite well with the main streets of Trondheim, as shown in figure 8.1, and

would fit many other city centers as well. Base stations are placed at regular intervals at some of the bigger streets. The chosen layout affects the simulation results. One characteristic of a grid layout with equal costs for each link is that there are often many paths to a destination with the same cost.

The layout models the location of base stations, not the location of end users. It is assumed that the end user equipment will select the base station it connects to automatically. As the user moves around, his or her equipment will switch base stations as necessary. Thus, base stations will transfer fluctuating amounts of data, depending on what traffic users generate and how they move around. The generated traffic depends on the services that users use and how those services are used. Base stations are connected by point-to-point links with a delay of $500ns$ between base stations. This value has been chosen because standard Cat5 ethernet cables have a delay of about $5ns/m$, resulting in a simulated distance between base stations of about 100 meter. How users are connected to the base station does not infuence the results and has not been specified in the simulations. Given the scenario, a wireless connection should be expected. It is assumed that users are connected to at most one base station at a time, which is true for standard WiFi connections.

All links between base stations are simulated at a bandwidth of $1Mbps$. That is a lower bandwidth than can be expected in reality. The low bandwidth is used to limit the number of packets that has to be simulated to congest a link. With higher bandwidths, the number of packets that has to be simulated grows, and so does the simulation time in ns-3. The effect can be limited by using bigger packets, but that has limitations as well: UDP packets have a size limit, which means that packets have to be split and the total number of packets grows again. It has been tested that a link of $1Mbps$ provides more than enough bandwidth to make the effect of CEAS packets on available bandwidth disappear completely: routing traffic uses a few kilobits per second at the very most, or less than one percent of available bandwidth.

Instead of simulating every end user, simulations are limited to simulating the network of base stations. The movement of users is simulated by changing the amount of traffic each base station generates. That way, the processing power needed for the simulation can be limited. Because end users are assumed to send all data to the base stations (i.e. no other networks are involved), the simulation should give the same results as a simulation including all the end users would.

As the user moves, different base stations are used as access points to the network. The trail that is followed is shown in figure 8.1. Every $1000s$, the user moves to the next base station. From $t = 8000s$, when the user is back at node 8, no more movements are made.

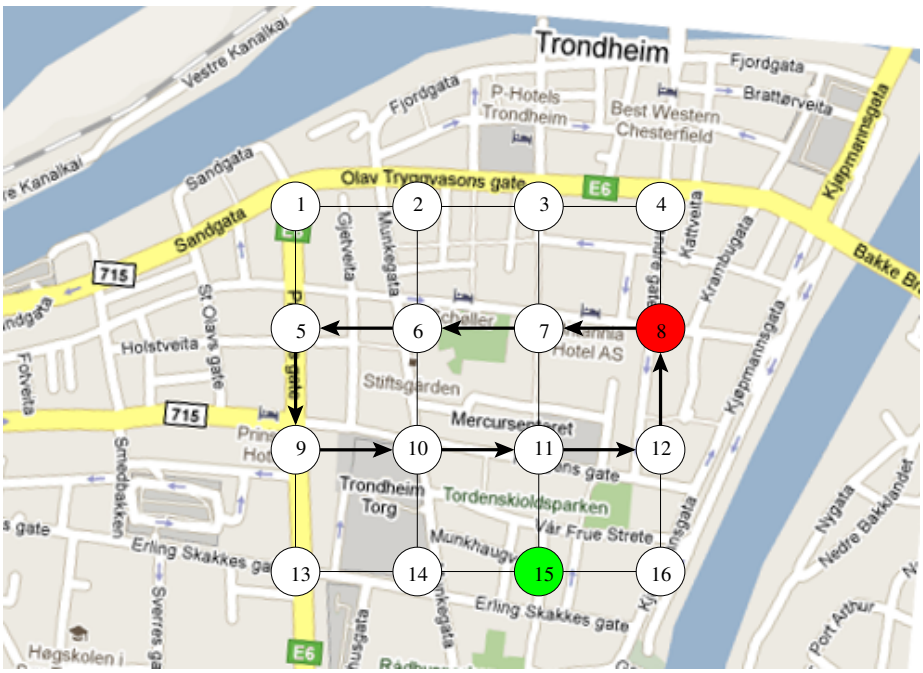


Figure 8.1: Trail through 16-node grid network. The red node is the start position of the traffic source. The source then follows the arrows to different nodes, moving every $1000s$. The green node is the destination of all generated traffic.

8.3 Basic results

The first simulations test how the system behaves when a user moves through the system.

In this simulation, no background traffic is generated at all and every base station has to start looking for a route to the destination when the user starts to use that base station. Once a destination has been requested, the base station will maintain the route - even if it is not requested again. As such, it is not completely realistic yet, but it provides a good way to see whether the scenario shows the expected behaviour. In this case, the expected behaviour is that the delay to the destination is directly proportional to the distance in number of hops to that destination. Also, the delay is expected to show a spike when the user arrives at a new base station. That spike comes from the time it takes to converge to the shortest route.

As shown in figure 8.2, the plot shows the expected behaviour: it shows spikes every time the traffic source - the user - moves to a new base station. The spikes are not clearly visible for those steps where the delay gets lower, because they are very narrow and overlap with the 'drop' from moving to a node which is closer to the destination. The measured delay is the delay of data packets

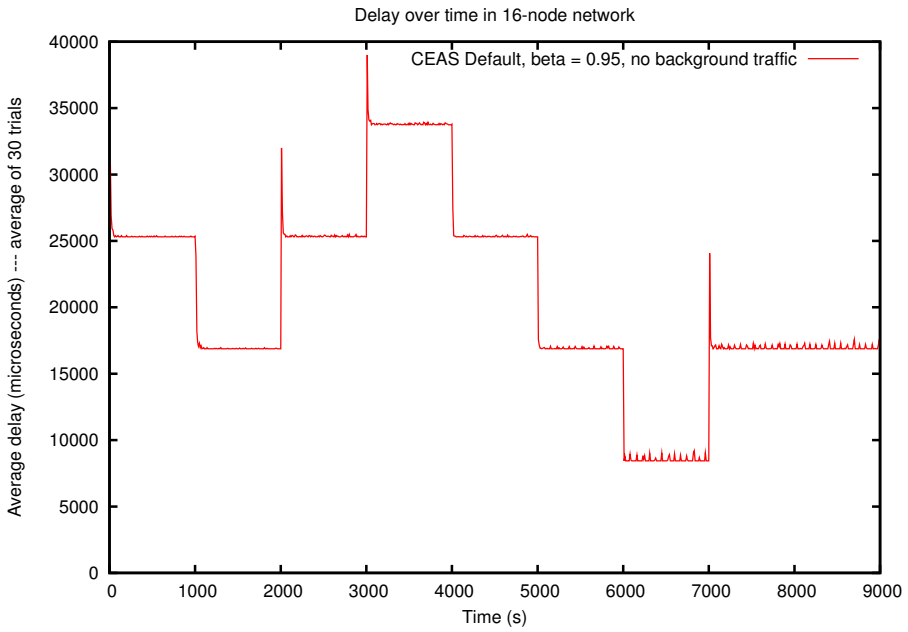


Figure 8.2: Delay over time - basic scenario.

and does not include routing packets. The parameters for the simulation are described in appendix A.

The next step is to include background traffic. The background traffic is a traffic flow which uses a significant part of the bandwidth along the dashed line in figure 8.3. The background traffic is enabled at $t = 3000s$. From then on, some paths to the destination are loaded and have a higher queueing delay. At all times, there is a route which is fast (in number of hops) and not loaded - except for $6000s \leq t < 7000s$, when the single shortest route is loaded.

The original implementation of CEAS in ns-2 uses hop count as the cost function. The ns-3 implementation uses that as the default as well. Because the cost function is not load-sensitive, the protocol can converge to a route which is heavily loaded and it will not switch to a different route until the path is so congested, that ants get dropped and the pheromone value for the path is not updated anymore. By then, a lot of data packets will have been dropped as well. The plot in figure 8.4 shows that the delay indeed grows dramatically when the path to which the protocol converges is congested. The line seems curved because of the logarithmic scale on the y-axis. In fact, it is linearly increasing. That is caused by the queues at the overloaded links: new packets arrive faster than they can be transferred, resulting in an unstable network. Note the drop in the average delay slightly after $t = 7000s$: the traffic source moves at that time, and at its destination it will take some time to converge to the shortest route. That means that longer routes are used for a moment as well, taking some pressure off the - overloaded - shortest route and resulting in a lower average delay. As soon as the system converges to the short route again,

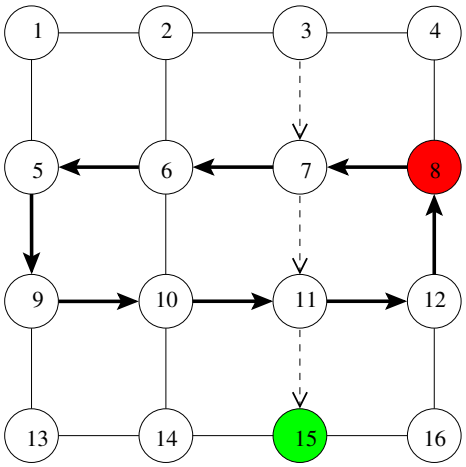


Figure 8.3: Trail through 16-node grid network with background traffic along the dashed line. Again, the red node is the start position of the traffic source and the green node is the destination of all generated traffic.

the link is again overloaded and the average delay will return to its old value and grow again.

8.4 Load-sensitive cost functions

To improve the situation with loaded links, a cost function which is load-sensitive has been added to the protocol. To be able to do that, the load of a link is estimated. The estimation formula used is:

$$\begin{aligned} \text{newTrafficRate} &= \text{smooth} \times \text{oldTrafficRate} + \\ &\quad (1.0 - \text{smooth}) \times \text{currentTrafficRate} \end{aligned}$$

The current traffic rate is estimated by $\frac{\text{receivedBits}}{\text{interval}}$, where *receivedBits* is the number of bits sent to the queue of the networking device. The interval is currently hardcoded to 10 seconds and the smoothing value is 0.5, which gives a reasonable estimation of the current outgoing traffic rate, without overreacting when relatively big packets are used. The parameters have been selected by running simulations with a few different values and looking for a combination that reacts quickly enough to changes in the traffic rate, yet does not give estimations that are too far from reality. That could happen when big packets are used. In that case, it is possible that there is less than one packet per second, which would result in quickly changing traffic rates that do not reflect the reality very well. A possible improvement would be to use the amount of data that has left the queue and has actually been sent over the link.

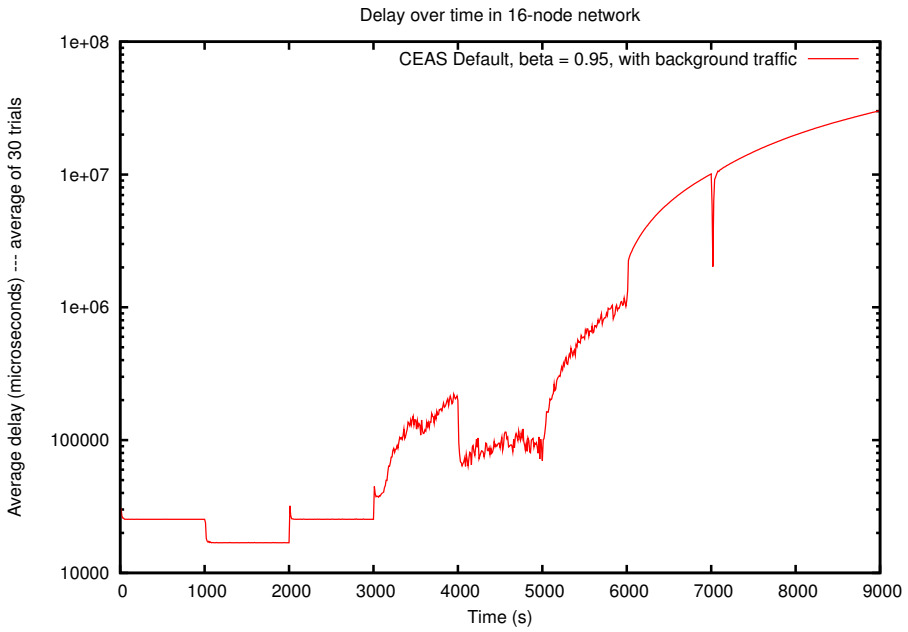


Figure 8.4: Delay over time - with background traffic from $t = 3000s$. The delay of the background traffic itself is not included in the calculation. Note the logarithmic scale of the y-axis: the delay does in fact grow linearly.

Without the smoothing mechanism, traffic rates could easily seem to be higher than the link capacity, simply because a few big packets were added to the queue in a short period. An extreme example would be a $50kbit$ packet, which is queued every 10 seconds on a $10kbps$ link. Clearly, the link is just used at half of its capacity - but without the smoothing parameter, the link would seem overloaded once every ten seconds and completely idle for the other nine seconds. Though seemingly unrealistic, similar situations actually occured during simulations with CEAS.

Implementing this estimation uncovered bugs in ns-3: the functions that were supposed to return the number of queued bytes and packets did not work properly. The problem has been solved; patches have been sent to the developers and have been added to the newest release of ns-3[1].

Given the estimation of the traffic rate, the load ρ is defined as $\frac{\text{traffic rate (bits/s)}}{\text{link capacity (bits/s)}}$, or $\frac{\lambda}{\mu}$ in variables from queueing theory. Note that this is a different ρ than the parameter used in CEAS.

The first load-sensitive cost function is $\frac{1}{1-\rho}$, which has the properties that are desired in such a function: it returns a low cost when the load is low and it grows as the load gets higher. In this case, the cost goes to infinity as the load reaches the link capacity.

[1] See http://mailman.isi.edu/pipermail/ns-developers/2009-December/007144.html and http://www.nsnam.org/bugzilla/show_bug.cgi?id=769.

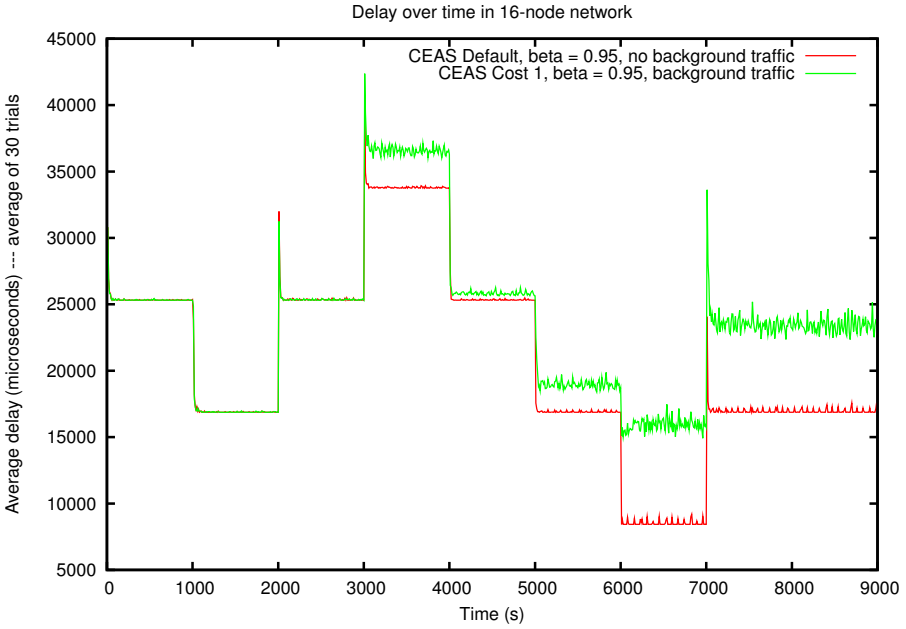


Figure 8.5: Delay over time with load-sensitive cost function. For comparison, the normal CEAS without background traffic has been included as well.

The resulting plot of using such a cost function can be seen in figure 8.5. When there are no congested links, the delay converges to the lowest possible delay. From $t = 3000s$, when some of the links are congested, part of the traffic is sent along a different path, resulting in a slightly higher end-to-end delay - but nothing like the delays seen with hop count as the cost metric.

Based on this result, simulations have been run with similar functions, but with a steeper curve - like $\frac{1}{(1-\rho)^2}$. The goal was to have a function which would react even faster on overloaded links by having a bigger difference in cost between loaded and unloaded links. However, those functions did not perform better than the cost function used in figure 8.5.

A different way of assigning a cost to a certain load is the following:

$$\text{cost} = \frac{\rho}{\text{totalBandwidth} - \text{usedBandwidth}}$$

With $\rho = \frac{\text{usedBandwidth}}{\text{totalBandwidth}} = \frac{\lambda}{\mu}$. A plot of both functions is shown in figure 8.6. This second function is similar to the first cost function, but it does not have an extra, constant cost. The idea is that the removal of the constant cost, which acts like an added cost to every hop, will make the system converge to the route with the lowest load. With the first function, a route of two hops would at least have a total cost of 2, even if both links are not loaded at all. A different route, which has length 1, will have a lower cost, even if it's slightly loaded.

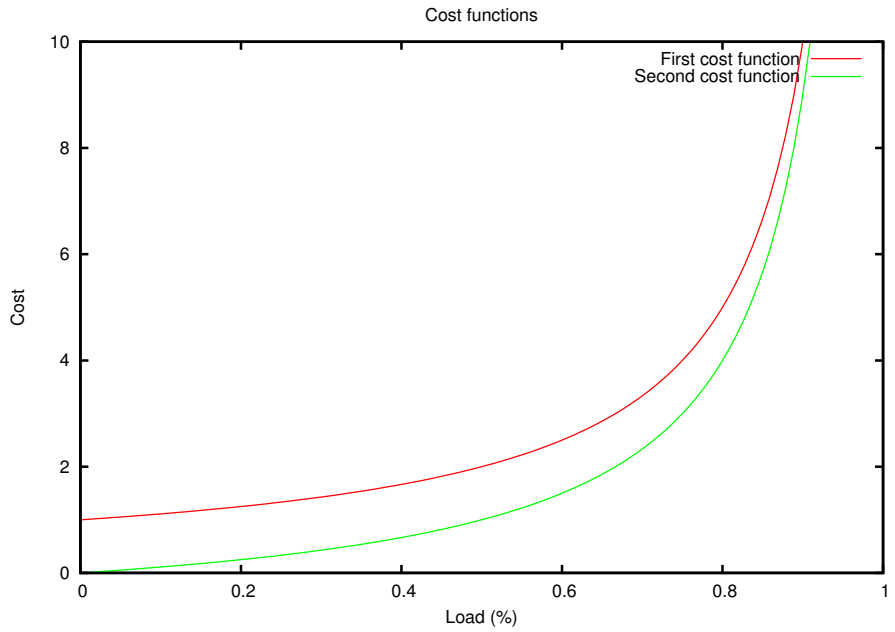


Figure 8.6: Cost as a function of link load for the first two cost functions.

Thus, the system would converge to the loaded link. In practice, that might not be bad, because the loaded link (given the low cost of 2) would still have a lot of capacity left - but for complete avoidance of loaded links, the second cost function would seem like a good candidate.

The resulting plot, shown in figure 8.7 shows that the second load-sensitive cost function does work, but in an unintended way. The relative cost difference between two paths at a low load is already high, which causes the system to keep switching to the lowest loaded path it can find, without taking the length into account. Thus, the system does not converge to the shortest path. The advantage is that an overloaded link will not change the end-to-end delay significantly: because the traffic is always balanced over different paths, only part of the traffic will be affected and the average delay will not change much. Of course, that's not very useful if that average delay is higher than the maximum delay that other cost functions return. The second cost function is not an improvement over the first function.

A third cost function uses λ and μ to estimate the mean waiting time and uses that as a cost value:

$$\text{cost} = \frac{1}{\text{totalBandwidth} - \text{availableBandwidth}} = \frac{1}{\mu - \lambda}$$

The cost value is the mean waiting time for an $M/M/1$ queue. Assuming that the arriving packets and their processing time follow a Poisson distribution, it

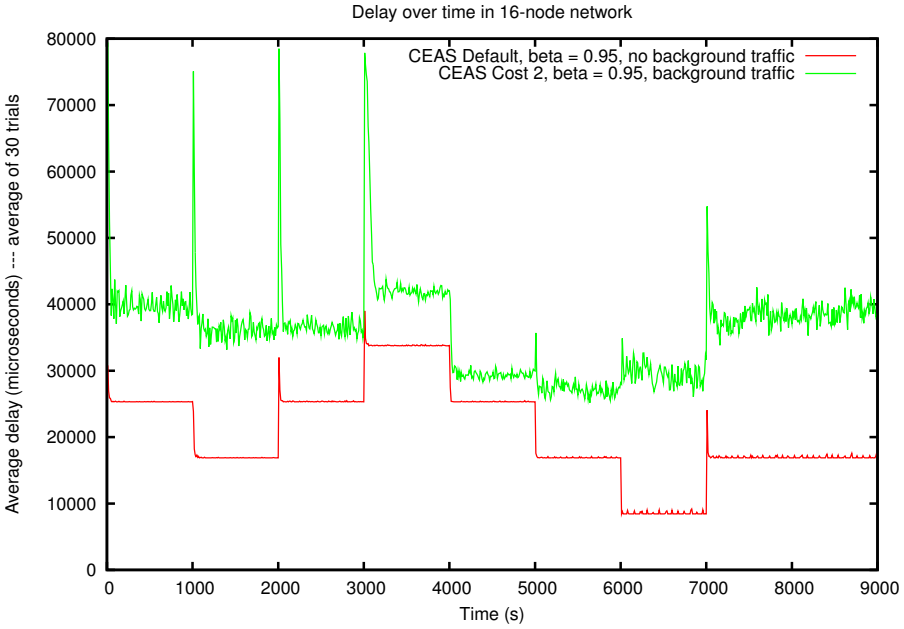


Figure 8.7: Delay over time with second load-sensitive cost function. For comparison, the normal CEAS without background traffic has been included as well.

should be a reasonable estimate of the expected waiting time in the real world.

As shown in figure 8.8, the third cost function performs very similar to the first cost function. In fact, it is not just 'very similar': the results of the two cost functions are identical. It is not difficult to see why that happens. The first cost function is:

$$\frac{1}{1-\rho}$$

That cost function can be shown to be equal to the third cost function, except for a constant factor:

$$\frac{1}{1-\rho} = \frac{1}{\mu - \lambda}$$

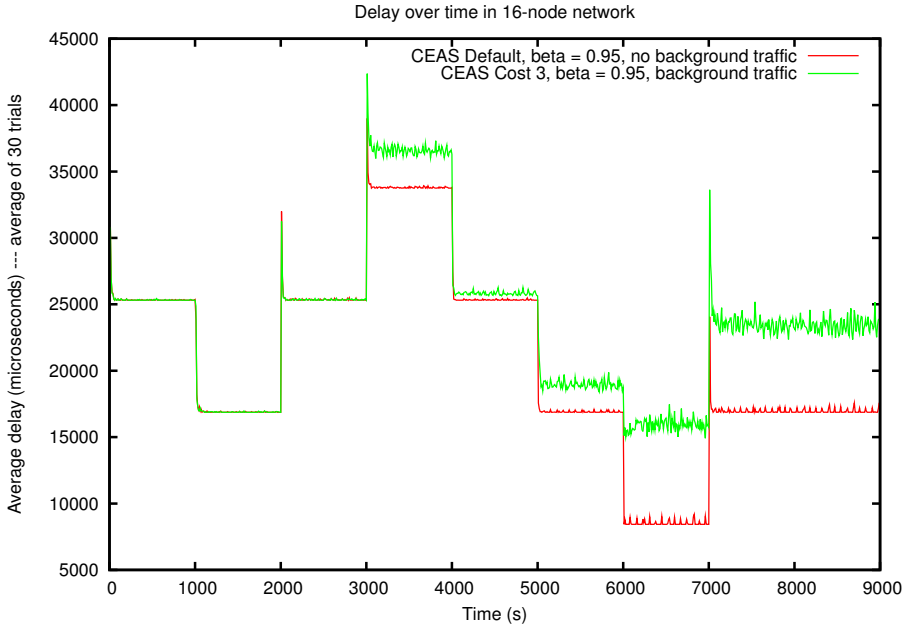


Figure 8.8: Delay over time with third load-sensitive cost function. For comparison, the normal CEAS without background traffic has been included as well.

$$\begin{aligned}
\frac{1}{1-\rho} &= \frac{1}{1-\frac{\lambda}{\mu}} \\
&= \frac{\mu}{\mu} * \frac{1}{1-\frac{\lambda}{\mu}} \\
&= \mu * \frac{1}{1-(\frac{\lambda}{\mu}) * \mu} \\
&= \mu * \frac{1}{\mu - \lambda}
\end{aligned}$$

As μ, which represents the total capacity (i.e. the service rate) of the link, is a constant value, the difference between cost function one and three is a constant factor. That implies that the ratios in pheromone values between different possible next hops is equal for both cost functions, which means that the same routing decisions will be taken. Thus, it does not make a difference whether the first cost function is used or the third function.

8.5 Preplanning

In the simulations so far, the routing protocol has been purely reactive: a node only starts to look for a route when there is a direct demand for it from a traffic source on that node. From then on, it will keep the route updated indefinitely.

Another approach is to use some type of proactive approach: routes are already discovered before they are requested. The advantage is that, when a request comes up, it should take less time - or even no time at all - to converge to the shortest route. The clear disadvantage is the extra overhead: ants will be sent to discover routes that will never be used. Whether the faster convergence is desirable enough to outweigh the overhead depends on the situation. In relatively static networks with many traffic sources, it makes sense to discover routes proactively: they would have to be discovered at a later time anyway, so it makes sense to do it in time and prevent unnecessary delays. In more dynamic networks with less traffic sources, it is probably not a good idea, because the chance that a route will actually be needed is smaller. On top of that, there is a bigger chance that a route which has been discovered before is not available anymore due to changes in the network topology: in that case, the 'investment' of finding a route early is lost and there will be a delay to find a new, working route anyway.

The CEAS implementation in ns-3 makes it possible to use preplanning. One approach is to discover routes to those hosts that any 'passing' data traffic has as its destination. The idea is that those destinations are relatively likely to be requested later on anyway, so it might pay off to look for routes to them. The results for preplanning those routes is shown in figure 8.9. As can be seen, the peaks that were visible when a user moved to a new base station have disappeared. That is the expected effect: routes have already been discovered when the user arrives, so no time is needed to converge to the shortest route.

8.6 Packet format improvements

The CEAS packet format as described in section 5.4, especially figure 5.3 and table 5.1, makes it easy to add new features to the protocol and test extensions such as subpaths (see subsection 2.2.5). However, it is not optimized for efficiency. A simple potential improvement to the protocol would be to use a more compact packet format. A suggestion for such a format is shown in table 8.1. Note that, contrary to the improvements mentioned in the last two sections, no simulations have been run with the compact format.

The encoding for each hop can be done as described in table 8.2.

Using the suggested compact format, a packet which has travelled for five hops would shrink from 113 bytes to 29 bytes. Some extra improvement would be possible by having distinct forward and backward ants, each with a separate

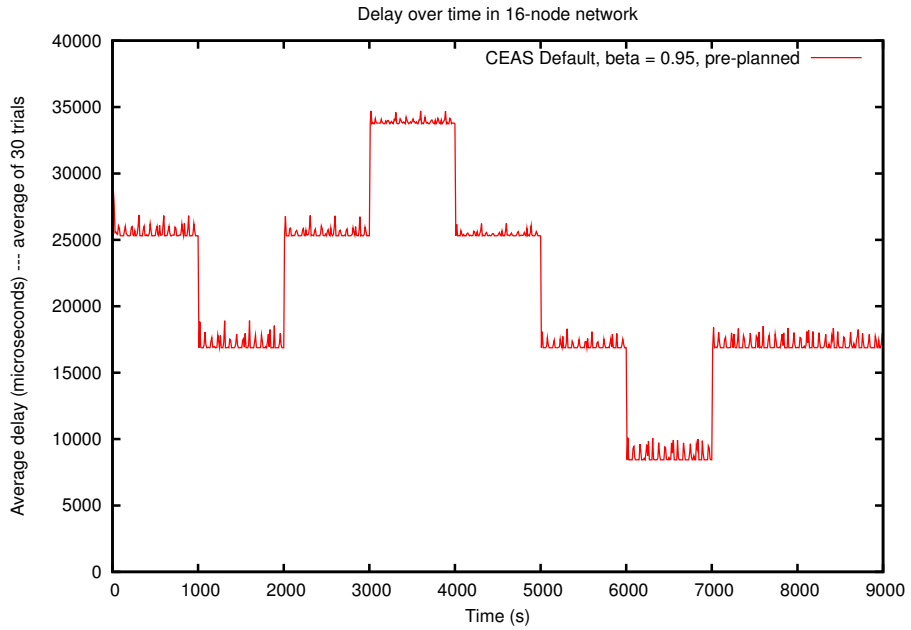


Figure 8.9: Delay over time with preplanning. The peaks when moving to a different base station, seen in simulations without preplanning, have disappeared.

Field	**Normal size (bytes)**	**Compacted size (bytes)**
Size	2	0
Ant type	1	0
TTL	1	1 (or a few bits)
Destination	4	4
Gamma	8	1
L	8	1
Rho	8	1
Number of hops	1	1
Total size	**33**	**9**

Table 8.1: Compact packet structure - changed fields in italic.

Field	**Normal size (bytes)**	**Compacted size (bytes)**
Source address	4	4
Destination address	4	0
Cost	8	0
Total size	**16**	**4**

Table 8.2: Packet structure - hops

structure. In that case, a value like γ would not have to be stored in a forward ant, resulting in an even smaller forward ant packet.

In the scenarios simulated in this report, the smaller packets will not make a significant difference: the link capacity and speed of any modern link will not show any measurable difference between packets of 31 or 123 bytes. Only in very specific situations, the difference might become significant. Such situations would include networks with narrow or very unstable links: in that case, smaller packets might have a higher chance of being transmitted successfully and a lower chance of causing congestion. In most cases, limiting the amount of ants will be much more effective than

The disadvantage of the more compact packet format is the loss of some possible extensions. As an example, the subpath extension depends on having the cost of every individual hop in the packet - which is not available in the compact format.

Chapter 9

Conclusion

9.1 The original plan

The goal of this thesis project, as stated at the start of it, was to extend CEAS for use in dynamic networks, in which the network topology or other circumstances change regularly. Would it be possible to adapt CEAS for such environments? And if so, what would have to be changed? Those changes would then be simulated to verify that the system would indeed work in dynamic environments.

Already at the beginning of the project, it was decided to implement CEAS in ns-3, instead of using the existing ns-2 implementation. The idea was that ns-3 would be easier to learn and to extend - resulting in faster improvements to CEAS once the implementation would be complete. Thus, the plan became to:

- Implement CEAS in ns-3
- Verify the implementation against the existing ns-2 implementation
- Come up with improvements to CEAS for use in environments with a relatively high rate of changes in the network topology
- Simulate the improvements to verify them

9.2 What actually happened

As the project progressed, it became clear that implementing the protocol in ns-3 took more time than expected. The implementation, which was originally intended to be a small part of the project, mainly to be able to improve things

faster later on, became a more important part of it. The first two items of the list above have been realized: CEAS has been implemented in ns-3 and the implementation has been verified.

The other two items, related to the improvement of CEAS, have been done to a smaller extend. Improvements have been proposed for one specific aspect of dynamicity in networks: congestion. The protocol has been adapted to behave in a better way in congested networks, while still showing good behaviour in networks where congestion is no issue.

Though the last items on the list have not been investigated as deeply as planned, some extra work has been done instead. Having CEAS implemented in both ns-2 and ns-3 allowed for a comparison between the two simulators. As it looks likely that both ns-2 and ns-3 will be used for some time to come, the comparison is interesting for anyone having to decide which simulator to use for his or her research. Together with Laurent Paquereau and Poul Heegaard, a paper has been written that describes the experience with both simulators. Our findings have been presented at a PhD course at NTNU and will be presented at SIMUL 2010.

Experience with ns-3

From the experience gained by implementing CEAS, ns-3 is a promising tool. It relies on good programming practices and provides carefully designed generic building blocks which make it easily extensible. Although some APIs are not fully stable yet, ns-3 is in many respects ready for active use and is to replace ns-2 as more and more models are added to it. The development of a framework for data collection and experiment control for ns-3 will also be a strong argument for moving to ns-3. One challenge remains maintaining the documentation and improving its coverage.

Whether to already take ns-3 in use depends on several criteria, including earlier experience with ns-2, availability of the models of interest and the type and scope of study.

For the time being and in the short term, ns-2 offers a larger number of models. Moreover, from our experience, porting a model from ns-2 to ns-3 is not trivial. However, the amount of time and effort to implement a new protocol in either of the tools is similar.

Compared to ns-2, ns-3 has a higher level of realism. Whether that increased realism is worth the added complexity depends on the situation. The rationale is to narrow the gap between simulation and a real implementation, and thus facilitate validation and emulation. To a certain extent, ns-3 is more of an emulator than a simulator. The downsides are twofold. First, ns-3 confronts the developer with low-level details at an early stage of the implementation. Hence, it misses the simplicity of ns-2 to quickly test research ideas, which is part of the reason for its popularity. Second, including more details leads to a

significant increase of the simulation run-time.

9.3 Future work

Now that ns-3 is stabilizing and a CEAS implementation for it exists, the next step would be to use the implementation to improve CEAS. That does not necessarily have to be for use in dynamic environments: the implementation allows for other types of research as well. Comparisons with other protocols should be relatively easy to make, as more and more protocols get implemented in ns-3.

As for improvements to CEAS, a first step would be to implement the subpath extension to the protocol in ns-3. This extension has been developed for ns-2 already and it has been shown to increase the performance of the protocol by significantly decreasing the number of ants that are needed to converge to good routes. From there, other changes could be made to use even less ants to converge - or to converge even faster with the same number of ants. Clearly, limiting the number of ants needed to react to changes in the network is especially important in networks where the rate of such changes is high. The ns-3 implementation should provide a good basis to implement those improvements and to simulate the results.

Appendix A

Simulation parameters

To make it possible to reproduce the results obtained, this report includes the parameters used to obtain the simulation results shown. These are the default values: in some cases, other values have been used. In that case, it is stated in the report.

Parameter	Value
Link bandwidth	$1Mbps$
Link delay	$500ns$
β	0.95
ρ	0.01
Ant rate	$1/s$ (constant intervals)
Explorer ant rate	$0.1/s$ (constant intervals)
Handling of loops	Described in section 5.5.3
Path cost	Sum of link cost on the way forward
Initial phase duration	$10s$
Elite selection	On; non-explorer ants only
Extra processing delay at nodes	0
Duration	$10000s$
Ant generation start	$t = 0s$
Number of runs	30
RNG seed	1234

Table A.1: Simulation parameters

Appendix B

Paper

The following paper has been written as part of the thesis project, together with Laurent Paquereau and Poul Heegaard. It has been accepted at SIMUL 2010[1].

[1] http://www.iaria.org/conferences2010/SIMUL10.html

Experience Report on Implementing and Simulating a Routing Protocol in NS-2 and NS-3

Jonathan Brugge[†], Laurent Paquereau[‡] and Poul E. Heegaard[†]
[†]Departement of Telematics
Norwegian University of Science and Technology, Trondheim, Norway
Email: brugge@stud.ntnu.no, poul.heegaard@item.ntnu.no
[‡]Centre for Quantifiable Quality of Service in Communication Systems*
Norwegian University of Science and Technology, Trondheim, Norway
Email: laurent.paquereau@q2s.ntnu.no

***Abstract*—Among specialized tools for simulation of network protocols, the network simulator 2 (ns-2) has been a first choice for over a decade. Its intended successor, ns-3, has been under active development since 2006 and was first made publicly available in 2008. However, ns-3 is not a new version of ns-2, but a completely new tool, and today ns-2 and ns-3 co-exist. In this paper, we report our practical experience in implementing and simulating the same routing protocol in both tools and, based on this experience, compare ns-2 and ns-3. In particular, we discuss the advantages and disadvantages of the increased realism of ns-3.**

***Keywords*-ns-2, ns-3, CEAS.**

I. Introduction

Simulation is an important method for the design, the evaluation and the presentation of network protocols. It allows to perform reproducible experiments over a wide range of scenarios and parameter settings. When conducting a simulation study, the choice of the simulation software is essential. There are two levels of choice. The first is the decision whether to use a general purpose programming language, a simulation library providing generic building blocks for implementing a simulator, or a specialized simulation tool including components and models that can be reused. The second is which language, library or specialized tool to use. In the first two cases, the developer's experience and proficiency in the language usually determinate the final choice. In the case of a specialized simulation tool, the choice should depend on many more criteria related to the tool itself, e.g. the abstraction level and the available models and components, but also to the type, objective and scope of the study.

Among specialized tools for simulating layer 2-4 network protocols, the *network simulator 2, ns-2* [1], has been the first choice for over a decade [2], [3]. The development of the next generation network simulator, *ns-3* [4], was initiated in 2006 [5] and the first version was released in 2008. Although it intends to eventually replace ns-2, ns-3 is not a new version of ns-2, but a complete overhaul, a new tool, and today ns-2 and ns-3 co-exist[1]. Hence, the question: ns-2 or ns-3? The objective of this paper is not to give any definite answer to that question, but rather to report our practical experience in implementing and simulating a routing protocol, namely the Cross-Entropy Ant System (CEAS), in both tools and, based on this experience, provide elements to help in the choice.

The rest of the paper is organized as follows. Section II provides background on CEAS, its applications and existing implementations. Sections III and IV detail the implementation and simulation of CEAS in ns-2 and ns-3, respectively. Next, ns-2 and ns-3 are compared in Section V. Finally, some concluding remarks are given in Section VI.

II. Background

A. CEAS

CEAS is a fully decentralized Ant Colony Optimization (ACO) system [6] first introduced in [7]. ACO systems are systems inspired by the foraging behaviour of ants in nature. ACO systems belong to the class of Swarm Intelligence (SI) systems [8]. SI systems are formed by a population of agents, whose behaviour is governed by a small set of simple rules and which, by their collective behaviour, are able to find good solutions to complex problems. ACO systems are characterized by the indirect communication between agents - ants - through local modifications of their environment, referred to as stigmergy and mediated by (artificial) pheromones. The pheromone trails reflect the knowledge acquired by the colony and good solutions emerge as the result of the iterative interactions between ants.

B. Applications and implementations of CEAS

CEAS was originally designed for distributed network path management and one of the successful application is adaptive routing in dynamic networks [9]. See [10] for a popular introduction to other applications of CEAS. Several different implementations of CEAS exist where the programming language and platform have been selected based on the objective of the study.

*"Centre for Quantifiable Quality of Service in Communication Systems, Centre of Excellence" appointed by The Research Council of Norway, funded by the Research Council, NTNU, UNINETT and Telenor. http://www.q2s.ntnu.no

[1]At the time of writing, the latest versions are 2.34 and 3.7, respectively.

CEAS has been mainly studied by simulation and most of the studies have been conducted using ns-2; see Section III. However, CEAS has also been implemented in DEMOS [11], a simulation library for SIMULA, to study the detailed behaviour of the system and its parameter sensitivity. In this implementation, the details of the communication protocol stack are abstracted to build a more efficient simulator.

CEAS has also been implemented in a testbed. Implementing a working prototype allows to validate simulation results. It also provides useful insights in the complexity of swarm-based methods in real routers and reveals potential implementation challenges and performance bottlenecks which are hard to predict through simulations and analysis alone. The first pioneering prototype implementation of CEAS [12] uses the Click Modular software [13] router system for packet forwarding and a Java-based Mobile Agent System called Kaariboga [14] for the routing process. An upgraded prototype implementation denoted AntPing [15] embeds new modules to Click for both CEAS routing and forwarding and is deployed on small home routers.

More recently, CEAS has been applied to the deployment of service components [16] and a SIMULA/DEMOS simulator has been developed for this purpose. To address CEAS in larger scale networks, PeerSim [17] is currently considered. PeerSim is a discrete event simulator designed for the simulation of systems with millions of nodes.

C. Application of CEAS to network path management

When CEAS is applied to network path management, ants are control packets used to repeatedly sample paths between source and destination pairs, and pheromone trail values are maintained locally at each node. For a given neighbour and destination, the pheromone trail value indicates the estimated goodness of the path using this neighbour as a next-hop towards the destination.

Fig. 1 illustrates the behaviour of ants in CEAS. Ants are generated by the source node and have a two-phase life cycle. On its way to the destination, a *forward ant* incrementally builds a path applying at each node a probabilistic forwarding decision based on the local pheromone trail values. Once it has reached the destination, an ant turns around and backtracks. On the way back, a *backward ant* deposits pheromones and triggers pheromone trail evaporation at each node along the sampled path. The path quality is evaluated at the destination where a self-adjusting parameter, denoted the temperature, is maintained. The temperature controls the relative weights given to solutions.

Moreover, additional mechanisms have been integrated to improve the performance of the system. The performance is measured in terms of quality of the solution, but also in number of iterations, time and overhead to converge or adapt after a change, e.g. a link failure. Elite selection is such a mechanism. It is applied at the destination and consists in discarding ants that have sampled low quality paths to reduce the overhead and improve the convergence speed. See for instance [9] for further details.

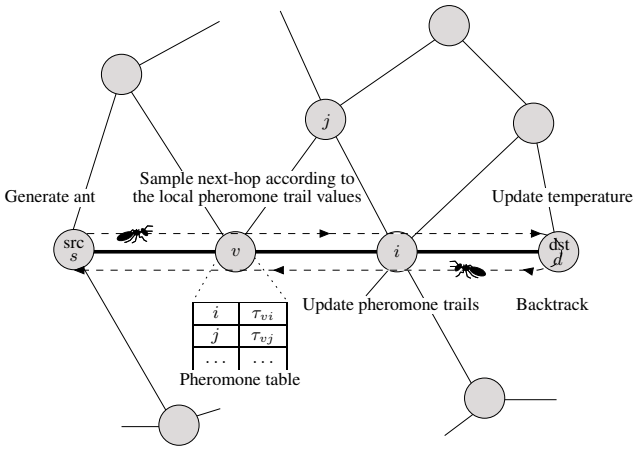


Fig. 1. Ant behaviour in CEAS

III. CEAS IN NS-2

A. Implementing CEAS

The first implementation of CEAS in ns-2 dates back to 2000 [7]. The choice of ns-2 at that time was motivated by a more general interest for network management by mobile software agents and the availability of the Active Network (AN) extension for ns-2 developed in the PANAMA project [18]. In this context, the support for Tcl scripts was also an important element. Moreover, although ant-based routing had already been proposed for several years, e.g. AntNet [19], no implementation was publicly available. Later, following the development of ns-2 and in an effort to better integrate CEAS, the implementation was revised according to the guidelines for implementing detailed dynamic routing protocols[2] given in the ns-2 documentation [20]. CEAS basic functions were then organized into `RtModule` (routing), `rtProto/Agent` (ant generation) and `Classifier` (forwarding). Finally, the latest version is based on the framework presented in [21] and implemented as a dynamic library for ns-2. The main rationale for this iteration was to make the implementation independent of the type and number of interfaces CEAS is running on.

The detailed structure of the latest implementation of CEAS in ns-2 is shown in Fig. 2. Arrows represent the path followed by packets internally. CEAS is implemented as a `NetworkLayerUnit`. Ants are defined as a new type of `Packet` and generated at the source node by a `CEASPacketGenerator`. At each node, ants are processed by a `CEASRoutingUnit`. Upon receiving a forward ant, the `CEASRoutingUnit` updates the ant, i.e. records the visit to the current node, and probabilistically chooses the next-hop to the destination based on the local pheromone trail values maintained in a `CEASPheromoneTable`. At the destination, the `CEASRoutingUnit` maintains CEAS internal parameters

[2]ns-2 distinguishes between *centralized routing protocols* and *detailed dynamic routing protocols*. The former refers to protocols running at the simulator level and computing routes having full-knowledge of the network; the latter to routing protocols running on each node, sending and receiving control packets and computing routes based on the local view of the network.

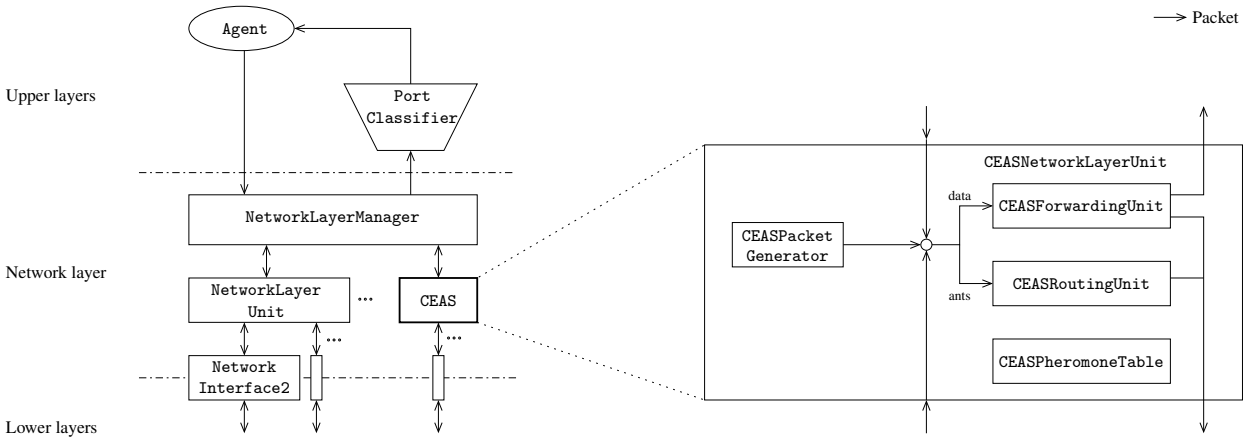


Fig. 2. Schematic representation of a `Node` [21] and detailed structure of the implementation of CEAS in ns-2

such as the temperature and performs additional operations including elite selection. Upon receiving a backward ant, the `CEASRoutingUnit` updates the local pheromone trail values. Data packet forwarding is handled by the `CEASForwardingUnit`. Incoming data packets are passed upwards to the `NetworkLayerManager`. Outgoing data packets are forwarded on a given interface based on the information available in the `CEASPheromoneTable`.

Other ant-based routing algorithms have been implemented in ns-2, e.g. AntNet [22] and AntSense [23]. Both protocols are implemented as routing `Agents` following the typical implementation of Mobile Ad-hoc NETwork (MANET) routing protocols in ns-2 [24] and, hence, inherit the limitations of this model [21]. For instance, in the case of AntNet, this design does not support the routing of data traffic.

B. Simulating CEAS

1) Tracing and data collection: The support for tracing and data collection in ns-2 is limited to recording events and basic statistics related to packets and triggered by packets passing through dedicated objects (`Traces` and `Monitors`, respectively) [20]. There is no native[3] support for tracing internal states such as the temperature values at the destination node or the pheromone trail values at intermediate nodes in the case of CEAS. This functionality therefore had to be integrated in addition to the implementation of the protocol itself. Moreover, ns-2 defines its own text format for trace records, so specialized scripts have to be written to post-process those traces.

2) Experiment control: ns-2 does not natively include tools for controlling an experiment[4], e.g. for running independent replications of the same scenario or running a simulation until a termination condition other than a specified time. In the case of CEAS, a relevant termination criterion may for instance be the convergence of the system. Hence, these features also had to be implemented.

[3]Extensions have been contributed, e.g. ns2measure [25], but are not included in the main distribution.

[4]Contributed extensions include ns-2/akaroa-2 [26] and ANSWER [27].

IV. CEAS IN NS-3

A. Implementing CEAS

CEAS is completely implemented within the frameworks of ns-3; see [28] for details. An overview of the architecture is shown in Fig. 3.

The main class is the `CeasRoutingProtocol`, which inherits from `Ipv4RoutingProtocol` and implements the protocol. There is one instance of such a protocol per node in the network. On receiving a forward ant, it is responsible for updating the ant and making the stochastic forwarding decision based on the local pheromone trails. At the destination, it maintains the temperature and applies elite selection. Finally, on receiving a backward ant, it is responsible for updating the local pheromone trails.

Pheromone trail values are stored at each node in a `CeasRoutingTable`. A `CeasRoutingTableEntry` contains a set of possible `NextHops`. `NextHop` wraps the routing table entry provided by ns-3 (`Ipv4Route`) and adds new fields, e.g. the associated pheromone trail value (`pheroLevel`) to reach the destination.

Ants are implemented as UDP datagrams[5]. Ant classes inherit from `Header` and implements serialization and deserialization methods. Internally, a `Packet` contains a serialized representation of `Headers`. When an ant is forwarded by the `CeasRoutingProtocol`, it is serialized into a `Packet`. When an ant is received, it is deserialized and handed over to the `CeasRoutingProtocol`.

When a data packet is to be forwarded, the transport protocol requests a route from `CeasRoutingProtocol` by calling `RouteOutput()`. When a data packet is received, the `Ipv4L3Protocol` calls `RouteInput()`; the `CeasRoutingProtocol` is then responsible for deciding whether the packet should be delivered locally or forwarded further.

[5]Contrary to the AntPing prototype implementation [15], the list of visited nodes is stored in the UDP payload and not using the IP route record extension. Hence, the number of hops is not limited to 8.

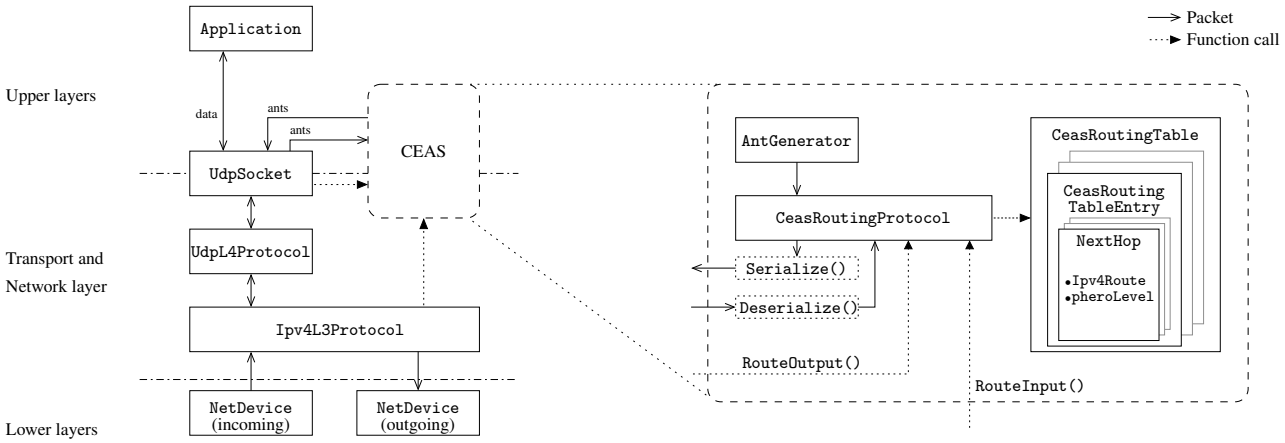


Fig. 3. Schematic representation of a `Node` and detailed structure of the implementation of CEAS in ns-3

B. Simulating CEAS

1) Tracing and data collection: Ns-3 supports packet tracing to ASCII (`AsciiWriter`) and pcap (`PcapWriter`) files. Specialized scripts have to be written to analyze ASCII files, while external tools such as Wireshark [29] can be used to examine pcap files. In addition, ns-3 provides a powerful framework for tracing internal variables (`TracedValues`) which implements the concept of independent trace sources and sinks and internally relies on callbacks. The support for data collection and statistics (`DataCollector`/`DataCalculator`), on the other hand, is much more limited. Output interfaces to OMNet++ [30] trace files and SQLite [31] are provided. Finally, a separate module allows to monitor packet flows (`FlowMonitor` [32]).

2) Experiment setup and control: ns-3 provides support for writing simulation scenarios. It includes `Helpers` to simplify the creation and the connection of objects and a comprehensive attribute system to access and configure objects. Attributes can be configured in the scenario itself or by passing arguments on the command line. Moreover, ns-3 includes experimental support for reading configuration from ASCII or XML files (`ConfigStore`). On the other hand, ns-3 does not provide support for controlling a simulation experiment. For instance, external scripts have to be written to run independent replications.

V. Comparing ns-2 and ns-3

This section highlights and discusses some of the differences between ns-2 and ns-3 as experienced during the implementation and simulation of CEAS.

A. Abstraction level

Both simulators are packet-based discrete-event simulators, but have different levels of abstraction.

Ns-3 mirrors real network components, protocols and APIs more closely. This becomes obvious when implementing CEAS in ns-2 and ns-3. At the transport and network layers, ns-3 does not abstract any detail. IP and UDP protocols are implemented in detail. A packet is represented as a buffer of bytes and the actual content of a packet needs to be serialized and deserialized. Packets are not simply sent: a socket has to be created and connected and errors have to be handled properly. Trying to connect to a closed port results in an ICMP error message. In ns-2, there is no detailed implementation of either UDP or IP.

The main reason and advantage of this increased realism is to facilitate code re-use, portability and validation. In particular, it makes it to possible to embed the simulator in a mixed environment with real hardware, software and networks. Any problem that would occur when implementing and running a protocol in a real-world system is likely to occur during the implementation and simulation in ns-3. The downside is the added complexity. Ns-3 confronts the developer with low-level implementation details such as socket communication, packet serialization and addressing at a very early stage. If one wants to try out novel concepts, such a level of details may be overwhelming. The higher level of abstraction in ns-2 allows for a quick implementation and testing of new ideas, and is an important reason for its popularity.

As a result, the implementation of the same protocol in ns-2 and ns-3 is significantly different and porting a protocol from ns-2 to ns-3 is not straightforward.

B. Usability and adaptability

By usability and adaptability, we mean how easy it is to learn the tool, extend existing models and add new ones. This involves many aspects:

1) Programming language and debugging: Ns-2 is implemented in C++ and OTcl. Each language taken separately is not difficult to use. The difficulty comes from the combination of the two and the concept of split-object. When developing a new protocol such as CEAS, one has not only to implement objects in both C++ and OTcl, but also the interactions between those objects. This task is made difficult by the lack of documentation and debugging tool for the interplay between C++ and OTcl. Ns-3 is written in C++ only and, hence, much easier to debug.

2) Building: Unlike ns-2, which uses the traditional GNU build system (autoconf, automake, make), ns-3 uses waf [33]. Waf is a much more recent framework, written in Python, that one needs to learn and adapt to when using ns-3.

3) Documentation: In addition to the ns-2 documentation [20], many tutorials and reports are available, e.g. [24]. However, not all modules are equally well documented and the documentation is in part outdated. Ns-3 is a much younger project and the amount of available resources is consequently much smaller. The development team strives to write and maintain a manual and tutorial as new models are integrated. Nevertheless, the coverage of the documentation is not complete yet, and parts have to be updated according to API changes.

4) Existing code: Ns-2 and ns-3 are open-source projects. Hence, a way to learn is to read and study existing code. Many models have been contributed to ns-2, but ns-2 code is generally hard to read because: (i) it includes old code for backward compatibility, (ii) many contributions use different coding styles and design approaches and constitute a patchwork of often incompatible models, and (iii) the code is generally poorly commented. In comparison, ns-3 enforces a coding style and a stricter review process before inclusion, which results in more coherent code which is better commented and easier to read. On the other hand, the number of examples is still limited. When we started to implement CEAS, the only example of dynamic detailed routing protocol was OLSR.

5) Software design: Both ns-2 and ns-3 follows an object-oriented design and are thus easily extensible. New modules are implemented as subclasses of generic base classes.

6) Modularity: In ns-2, it is not always easy to simply replace a model by another. In the case of routing protocols, models vary depending on the type and numbers of interfaces; see [21]. In ns-3, layers are clearly separated and interfaces well-defined. Replacing objects by similar ones, e.g. a routing protocol, is therefore much easier. On the other hand, the architecture of ns-3 closely maps that of existing systems and implementing untraditional approaches or different levels of abstraction, e.g. abstracting the IP layer, is much more demanding.

C. Experiment setup, control and analysis

Both tools provide basic network elements such as nodes and links and make it easy to setup simulation scenarios. In particular, ns-3 provides various helpers to facilitate the creation, initialization and connection of the different entities in the simulation. However, both simulators natively offer very limited support for data collection and experiment control. For instance, neither of them provide mechanisms for transient period detection, specification of termination conditions other than time, handling of replications, or parallel and distributed execution. For ns-2, several frameworks have been developed and provide some of these features. For ns-3, most of these features are being developed or planned, but not yet integrated.

Furthermore, compared with ns-2, ns-3 provides a powerful framework for tracing internal variables, but, for the time being, misses generic trace sinks. Other advantages of ns-3 include the use of standard formats, such as pcap for packet tracing, and the integration of interfaces to external software such as SQLite.

Finally, the support for visualization in ns-2 and ns-3 is limited (Nam and NetAnim, respectively), in particular for wireless networks.

D. Development status

Ns-2 is funded through the ns-3 project, but the core development team is only working on ns-3. Ns-2 only receives maintenance updates and less and less models are contributed. Ns-3, on the other hand, is under active development. It has been available for developers and early adopters since 2008. Most of the generic building blocks are in place. However, not all the core APIs are completely stable yet, which may keep some developers from moving to ns-3. For example, the routing API has undergone significant changes until version 3.6 (October 2009). One should also expect some rough edges. For instance, during the development of CEAS, it became clear to us that the SQLite output interface was a performance bottleneck and had to be fixed by introducing support for SQL transactions. The resulting patch has since then been integrated.

The original NSF project for ns-3 is ending this year (2010) and a lot has already been achieved. However, referring to the project goals [5], there is still much to do, including porting models from ns-2, providing support for data collection, experiment control and statistic generation, extending the visualization support, and integrating ns-3 with external tools such as Click. Recently, a new NSF grant has been announced for the development of "frameworks for ns-3" [34]. The framework will focus on better support for controlling the execution of simulations and analyzing the results. Finally, part of the development of ns-3 is also founded through the Google Summer of Code (GSoC) [35] program.

E. Efficiency

The performance of ns-2 and ns-3 was evaluated by simulating a relatively simple scenario, similar to the one used in [36]. In this scenario, the network is composed of ten nodes connected by point-to-point links. CEAS is used to find the shortest path between a source and destination pair. The results are given in Table I. The run-time and the number of events are averaged over 30 replications. The standard deviation is given in parentheses. In this case, ns-2 turned out to be almost ten times faster than ns-3. Profiling showed that a significant part of the difference comes from the time spent on serializing and deserializing packets and from the detailed implementation of the transport and network layers.

The efficiency of both simulators was also compared in terms of memory requirements. For this purpose, a scenario with 10.000 nodes containing an IP stack and the CEAS implementation was loaded with both simulator. In this test, ns-3 used half as much memory as ns-2. The difference comes from the fact that the ns-3 node is simply a container and only

TABLE I
PERFORMANCE COMPARISON

Simulator	Run time	Number of events
ns-2	0.94s (0.04)	157511 (6689)
ns-3	8.92s (0.33)	412104 (9098)

the required components are instantiated while the ns-2 node is a much more static construct including components that may not be used.

VI. CONCLUDING REMARKS

From our experience, ns-3 is a promising tool. It relies on good programming practices and provides carefully designed generic building blocks which make it easily extensible. Although some APIs are not fully stable yet, ns-3 is in many respects ready for active use and is to replace ns-2 as more and more models are added to it. The development of a framework for data collection and experiment control for ns-3 will also be a strong argument for moving to ns-3. One challenge remains maintaining the documentation and improving its coverage.

Whether to already take ns-3 in use depends on several criteria, including earlier experience with ns-2, availability of the models of interest and the type and scope of study.

For the time being and in the short term, ns-2 offers a larger number of models. Moreover, from our experience, porting a model from ns-2 to ns-3 is not trivial. However, the amount of time and effort to implement a new protocol in either of the tools is similar.

Compared to ns-2, ns-3 has a higher level of realism. Whether that increased realism is worth the added complexity depends on the situation. The rationale is to narrow the gap between simulation and a real implementation, and thus facilitate validation and emulation. To a certain extent, ns-3 is more of an emulator than a simulator. The downsides are twofold. First, ns-3 confronts the developer with low-level details at an early stage of the implementation. Hence, it misses the simplicity of ns-2 to quickly test research ideas, which is part of the reason for its popularity. Second, including more details leads to a significant increase of the simulation run-time.

REFERENCES

[1] The network simulator - ns-2. [Online]. Available: http://www.isi.edu/nsnam/ns

[2] S. Kurkowski, T. Camp, and M. Colagrosso, "MANET simulation studies: the incredibles," *SIGMOBILE Mobile Computing and Communications Review (MC^2R)*, vol. 9, no. 4, pp. 50–61, 2005.

[3] *ns-3 project description*. [Online]. Available: http://www.nsnam.org/docs/proposal/project.pdf

[4] The ns-3 network simulator. [Online]. Available: http://www.nsnam.org

[5] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley, "ns-3 project goals," in *Proc. First Workshop on NS2 (WNS2)*, Pisa, Italy, Oct. 2006.

[6] M. Dorigo, G. Di Caro, and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artificial Life*, vol. 5, no. 2, pp. 137–172, 1999.

[7] B. E. Helvik and O. J. Wittner, "Using the cross entropy method to guide/govern mobile agent's path finding in networks," in *Proc. International Workshop on Mobile Agents for Telecommunication Applications (MATA)*, Montreal, Canada, Aug. 2001.

[8] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artifical Systems*. Oxford University Press, 1999.

[9] P. E. Heegaard and O. J. Wittner, "Overhead reduction in distributed path management system," *Computer Networks*, vol. 54, no. 6, pp. 1019–1041, Apr. 2010.

[10] P. E. Heegaard, B. E. Helvik, and O. J. Wittner, *The Cross Entropy Ant System for Network Path Management*, ser. Telektronikk, 2008, vol. 104, no. 1, pp. 19–40.

[11] G. Birtwistle, "DEMOS - a system for Discrete Event Modelling On Simula," 1997. [Online]. Available: http://www.dcs.shef.ac.uk/~graham/research/demos.pdf

[12] A. Mykkeltveit, P. E. Heegaard, and O. J. Wittner, "Realization of a distributed route management system on software routers," in *Proc. Norsk Informatikkonferanse (NIK)*, Stavanger, Norway, Nov./Dec. 2004.

[13] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[14] D. Struve. (2006, Sep.) Kaariboga mobile agents. [Online]. Available: http://www.projectory.de/kaariboga

[15] P. E. Heegaard and I. Fuglem, "Demonstrator 1: Ant-based monitoring on software IP routers," BISON (IST-2001-38923), Tech. Rep., 2006. [Online]. Available: http://www.cs.unibo.it/bison/deliverables/D14.pdf

[16] M. J. Csorba, P. E. Heegaard, and P. Herrmann, "Component deployment using parallel ant-nests," *International Journal of Autonomous and Adaptive Communications Systems (IJAACS)*, 2010.

[17] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris. The Peersim simulator. [Online]. Available: http://peersim.sf.net

[18] O. J. Wittner and B. E. Helvik, "Simulating mobile agent based network management using network simulator," in *Poster abstracts Joint International Symposium on Agent Systems and Applications / Mobile Agents (ASA/MA)*, Zürich, Switzerland, Sep. 2000.

[19] G. Di Caro and M. Dorigo, "AntNet: Distributed stigmergetic control for communications networks," *Journal of Artificial Intelligence Research (JAIR)*, vol. 9, pp. 317–365, 1998.

[20] *The* ns *Manual*, Kevin Fall and Kannan Varadhan ed., The VINT Project, UC Berkeley, LBL, USC/ISI, and Xerox PARC.

[21] L. Paquereau and B. E. Helvik, "Simulation of wireless multi-* networks in ns-2," in *Proc. Workshop on NS2 (WNS2)*, Athens, Greece, Oct. 2008.

[22] V. Laxmi, L. Jain, and M. S. Gaur, "Ant colony optimisation based routing on ns-2," in *Proc. International Conference on Wireless Communication and Sensor Networks (WCSN)*, Allahabad, India, Dec. 2006.

[23] T. Camilo, C. Carreto, J. S. Silva, and F. Boavida, "An energy-efficient ant-based routing algorithm for wireless sensor networks," in *Proc. International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS)*, ser. LNCS, vol. 4150. Springer Berlin / Heidelberg, 2006.

[24] F. J. Ros and P. M. Ruiz, "Implementing a new manet unicast routing protocol in ns2," Dec. 2004. [Online]. Available: http://masimum.inf.um.es/nsrt-howto/pdf/nsrt-howto.pdf

[25] C. Cicconetti, E. Mingozzi, and G. Stea, "An integrated framework for enabling effective data collection and statistical analysis with ns-2," in *Proc. Workshop on NS2 (WNS2)*, Pisa, Italy, Oct. 2006.

[26] The ns-2/akaroa-2 project. [Online]. Available: http://www-tkn.ee.tu-berlin.de/research/ns-2_akaroa-2/ns.html

[27] M. M. Andreozzi, G. Stea, and C. Vallati, "A framework for large-scale simulations and output result analysis with ns-2," in *Proc. International Conference on Simulation Tools and Techniques (SIMUTools)*, Rome, Italy, Mar. 2009.

[28] *ns-3 reference manual*. [Online]. Available: http://www.nsnam.org/docs/release/manual.pdf

[29] Wireshark. [Online]. Available: http://www.wireshark.org

[30] OMNet++. [Online]. Available: http://www.omnetpp.org

[31] SQLite. [Online]. Available: http://www.sqlite.org

[32] G. Carneiro, P. Fortuna, and M. Ricardo, "FlowMonitor - a network monitoring framework for the network simulator 3 (ns-3)," in *Proc. International Workshop on Network Simulation Tools (NSTools)*, Pisa, Italy, Oct. 2009.

[33] Waf. [Online]. Available: http://code.google.com/p/waf

[34] NSF Grant "Frameworks for ns-3". [Online]. Available: http://nsf.gov/awardsearch/showAward.do?AwardNumber=0958015

[35] Google Summer of Code. [Online]. Available: http://code.google.com/soc

[36] V. Kjeldsen, O. J. Wittner, and P. E. Heegaard, "Distributed and scalable path management by a system of cooperating ants," in *Proc. International Conference on Communications in Computing (CIC)*, Las Vegas, NV, USA, Jul. 2008.

Appendix C

Presentation TM8105

Poul Heegaard organizes a course 'Advanced Discrete Event Simulation Methodology' for PhD students. During one lecture, both ns-2 and ns-3 were introduced to participants. Laurent Paquereau spoke about ns-2. The following slides show what I have presented about ns-3.

What is ns-3?

Discrete-event network simulator Focused on studying networking protocols in a controlled environment

Free, open source software project All code released under GPLv2 or compatible licences

Not ns-2 Ns-3 is a new simulator, not backwards-compatible with ns-2

More information can be found at *www.nsnam.org*.

History

Ns-3 got started because limitations in ns-2 could not be fixed easily.

July 2006 Official start of ns-3

March 2007 First development release (ns-3.0.1)

June 2008 First stable release (ns-3.1)

Since then Releases every 3-4 months

The current release is ns-3.6, released in October 2009. Version 3.7 is expected in early 2010.

Why does ns-3 exist?

Or: *what is wrong with ns-2?*

Architectural issues Core is difficult to extend and has scalability problems

Bi-language system (C++/tcl) The combination is difficult to debug and a barrier for new developers

Core packet structure Packet structure not suitable for emulation

Lack of validation and verification Many models in ns-2 are not validated against the real world

Fixing the above is impossible without breaking backward compatibility.

Not completely from scratch

Ns-3 is not a completely new simulator. It has taken code and concepts from:

- Ns-2 (OLSR, error models)
- Yans (WiFi model)
- GTNetS (applications)

Also, it builds on years of experience from ns-2.

Features (1/3)

- Extensible core
  - C++ with an optional Python interface
  - Documented more extensively than ns-2
  - Callbacks to limit coupling between models
- Flexible object aggregation
  - (covered later)
- Attention to realism
  - Models nodes like a real computer
  - Supports sockets API and other interfaces

Features (2/3)

- Software integration
  - Trace output can be in pcap format and used in Wireshark, for instance
  - Hooks to GNUplot, netanim, potentially more
- Support for virtualization and testbeds
  - Ns-3 can interact with 'real' systems
- Tracing and statistics
  - Trace sources (packet reception, etc.) are decoupled from trace sinks (output interfaces)
  - User can freely couple trace sources en sinks
  - Statistics module exists - though limited functionality so far

Features (3/3)

- Lots of smart pointers and templates
  - Not necessarily a feature, but good to know beforehand
  - *Ptr<Ipv4Route> route = CreateObject<Ipv4Route>();*
- Easy attribute configuration
  - Pointer-based: *txQueue->GetAttribute ("MaxPackets", limit);*
  - String-based: *Config::Set ("/NodeList/*/DeviceList/*/TxQueue/MaxPackets", UintegerValue (15));*
- Helper scripts
  - Make programming scenarios easier
  - (shown later)

Object aggregation - example

```
Ptr<Ipv4> ipv4 = CreateObject<Ipv4> ();
ipv4->SetNode (node);
node->AggregateObject (ipv4);

ipv4 = m_node->GetObject<Ipv4> ();
```

Instead of

```
Ipv4Node in = static_cast<Ipv4Node>(node);
Ipv4 ipv4 = in.GetIpv4();
```


Event schedulers

- In ns-2: Heap, List, Calendar, Splay, RealTime
- In ns-3: Heap, List, Calendar, Ns2Calendar, *Map* (default)

Real time scheduling is implemented as a different simulator object in ns-3, not as an event scheduler.

Network components and protocols

Network stacks ARP, IPv4, ICMPv4, UDP, TCP (IPv6 under review)

Devices WiFi, CSMA, point-to-point, bridge

Error models and queues

Applications UDP echo, on/off, sink

Mobility models Random walk, etc.

Routing OLSR, static global, AODV (soon), CEAS

⇒ Growing, but still less than in ns-2.

Packets in ns-3

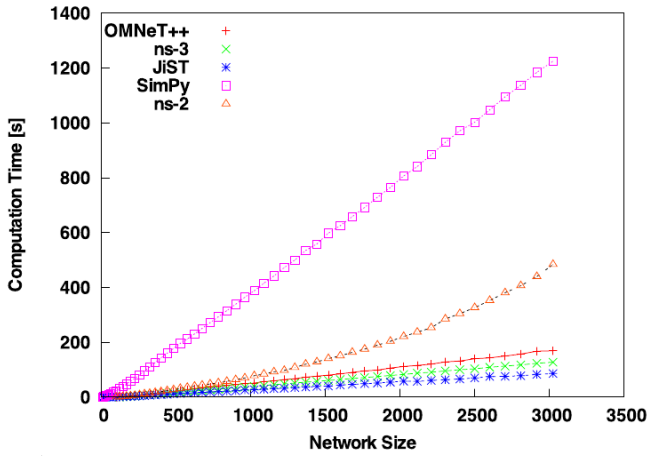
In ns-2, packets are modelled as events. In ns-3, packets are 'real' packets:

- Packets are serialized/deserialized
- Allows implementation of fragmentation
- Allows testing against 'real world'

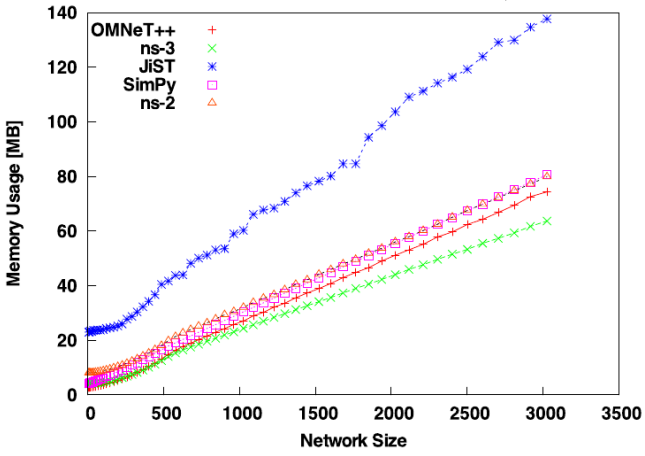
On the other hand:

- Requires more work from the user
- Is potentially slower

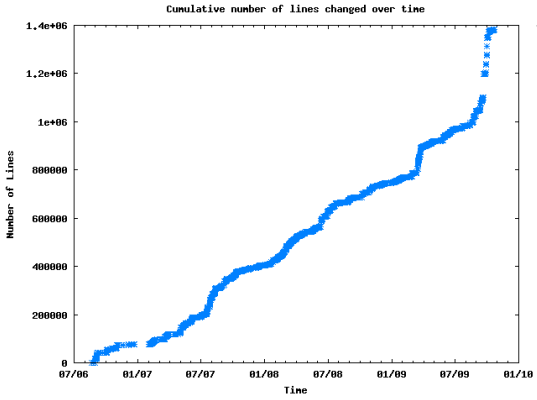
Performance - computation time



Performance - memory usage

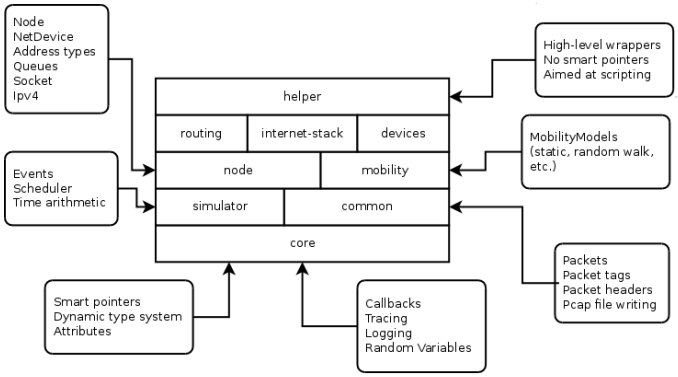


Actively developed



⇒ Size: ∼ 220.000 lines in main distribution

Architecture



A typical simulation

- Create (C++) objects
- Configure and connect them
- Schedule special events if needed
- Run the simulation

⇒ In practice, changes to the core are often still required.

Example scenario

```
Node *a = new Node ();
Node *b = new Node ();
Link *link = new Link (a,b);
Simulator::Schedule (Seconds (0.5),
                     &Node::StartCbr, a,
                     "100bytes", "0.2ms", b);
Simulator::Run ();
```


Demonstration

Demonstration of simple ns-3 scenario

Demonstration - discussion

- Demonstration is realistic, but not very useful
- Real simulation will require actual programming

My current workflow:

1. Shell script runs scenario number of times
2. Scenario uses my own code (routing protocol)
3. Scenario stores results in SQLite database
4. Shell script runs queries on the database
5. Shell script generates plot from the query results

State of ns-3

As it is now, ns-3 is ready for the first users:

- API is stabilizing, *but not stable yet*. Fundamental changes still happen.
- Code quality is good
- Most required pieces are in place
- Check whether pieces *you* require are there

Own experience (1/2)

Good stuff:

- Documentation (when it's there)
- Readable, structured source code
- Helpful and active developers
- Improving fast - new models added regularly

Could be better:

- Documentation (when it's not there)
- Non-standard build system (Python-based 'waf')
- Statistics infrastructure clearly unfinished
- Sometimes unexpected things are missing

Own experience (2/2)

- Realistic simulations are nice...
- ...but ns-3 still confronts users a bit too much with 'realism'
- Debugging still difficult (C++ and GDB)
- Not many people with ns-3 experience around (yet)

Use ns-2 or ns-3?

- At this point, added realism is both a blessing and a curse
- For instance, requires detailed knowledge of IP addresses and BSD sockets API.
- Ns-2 has more models, but ns-3 is developing much faster
- From own experience: ns-3 is usable for projects - but watch out for missing parts

Questions

Any questions?

Sources

- Performance graphs taken from *A performance comparison of recent network simulators* by Weingartner et al., 2009
- Lines-of-code graph taken from the frontpage of the ns-3 website (www.nsnam.org)
- Feature overview based on *ns-3 overview*, published on the ns-3 website

Appendix D

Research project

The research project is a project to prepare for the master thesis. As part of the information in this thesis builds on the research project, particularly in chapter 1 and 2, it is attached to this thesis.

Research Topics

Jonathan Brugge

December 19, 2009

Contents

Chapter 1

Introduction

This research project is a preparation for my master thesis at NTNU in Trondheim. Based on this report, a final thesis subject will be defined. A first idea is to write my thesis about adapting the Cross Entropy Ant System for use in dynamic networks. To be able to really define the subject, this report has two goals:

- Describe the Cross Entropy Ant System (CEAS)
- Give an overview of routing in dynamic networks

The combination of those two topics should provide a basis for the definition of the thesis subject.

To be able to describe CEAS, the first part of this report gives an introduction in some of the tools used in the design of CEAS. Based on that, CEAS is described. The second part of this report gives a number of design criteria for routing protocols in dynamic networks and continues to describe those protocols and how well they satisfy the design criteria.

Chapter 2

Basics

This chapter gives an introduction to the tools that are used in the following chapters. Rare event simulation is described and an explanation of importance sampling, a technique that is used in rare event simulation, is given. The information in this chapter is needed to understand the rest of this document.

2.1 Simulation

Sometimes, the behaviour of a system has to be analyzed. In this section, I'll use a slot machine to describe the various ways to perform such an analysis. We want to know how often a user gets the jackpot.

The easiest way to learn that number is to systematically try it: just toss in coins until you've hit the jackpot a couple of times and calculate the chance of getting it. While time intensive, this is a reliable way to learn more about the behaviour of the machine.

While easy, the hands-on approach does have disadvantages. Assuming the owner of the slot machine plans to make money, it can be an expensive way to test the machine. Also, when the configuration of the machine is changed in any way, all previous results are worthless and testing has to start again. Finally, it is not very precise if the chance of winning the jackpot is relatively small: if you use the slot machine 100 times and you hit the jackpot once, you still do not really know what the chance of winning the jackpot really is: you might have been lucky to hit it even if the real chance is more like one in a million. To get a better estimation, you'd have to win it very often - which becomes increasingly difficult with machines that have a small chance of winning.

Other systems might have more problems: while it is possible to use a slot machine many times in a row, that is more difficult with a montly lottery: it

would probably take hundreds of years before you'd have an idea about the chance to win.

The most precise way to analyze the behaviour of a model is to mathematically analyze it. While that gives a complete idea of the behaviour of the model, it is often difficult or even impossible. The calculations involved quickly become too difficult to solve analytically and even if it possible, it takes too much time. In case of the slot machine, modelling would not be very difficult: given the start position, speed and direction of each wheel in the slot machine and any other parameters that influence its behaviour, a reasonable model could be made. However, solving the equations you'd get when you assume a certain behaviour of the user of the machine (who influences the success rate as well, of course) would probably become unfeasible. Considering that slot machines are not exactly the most complicated devices imaginable, it is clear that analytically describing the behaviour is not a good solution for all problems.

An easier, though less precise method is to simulate the behaviour of the model. As said before, creating a model of a system can be feasible, even if it is difficult to analyze. If one wants to know how often a model reaches a specific state, the idea is to run the model as often as needed and check each time whether that state is reached. That process can be automated and thus be performed as often as needed: now, it is suddenly possible to play the slot machine a million times without spending any coins - and you'd get a better approximation of the chance of hitting the jackpot, because the large number of tests makes the uncertainty in the measured quantity smaller.

2.2 Rare event simulation

2.2.1 Introduction

The procedure described before works quite well, assuming the slot machine gives you a reasonable chance to win the jackpot - say, once every thousand plays or so. You'd just simulate a million games, get about a thousand times in the 'jackpot state' and conclude that the chance is about one in a thousand.

However, if the slot machine lets you only win once every million games, you'd have to play so many games that it would become difficult to simulate. The situation is even worse if you'd like to calculate, say, the chance of being hit by a meteorite in the next minute. Such events are really rare and special tools are needed to be able to simulate such situations. That is the area of *rare event simulation*.

The basic idea in rare event simulation is to change the system parameters in such a way that the rare event becomes a more common event: you amplify it. Then, you can simulate just as you'd do in the normal case and when you know how often the event occurs, you divide by the amplification factor and you know

how often the rare event would occur. This trick is called *importance sampling*.

2.2.2 Importance sampling

As a simple example, consider the meteorite example mentioned before. One could change various parameters:

- Increase the number of meteorites
- Increase the time from one minute to much longer
- Increase the size of the target, in this case the human.

Assume that the target is a billion times larger. The chance of being hit increases approximately a billionfold and thus simulation becomes doable. After one gets the result, a simple division by a billion yields the right answer.

This example also shows a problem that can occur with this approach. If, instead of the size of the target, the number of meteorites would have been increased, the chance of being hit would be a billion times larger as well. However, the lower number of simulation sessions required would not help all that much if the time to perform a single simulation experiment would significantly increase - which is not unlikely if there are a billion times more meteorites included in each simulation.

Of course, there is another problem as well. Increasing parameters does not necessarily linearly increase the chance that an event occurs and parameters might be related to each other. Consider the slot machine discussed before: it is not immediately apparent what exactly would have to be changed to increase the success rate tenfold. Determining the right change of parameters in the general case is a hard problem. Heuristics have been developed. One of those heuristics, called the *cross entropy method*, is described in the next section.

2.3 Cross entropy method

2.3.1 Introduction

Importance sampling only works if the parameters of the simulation are well chosen. One technique to do this is called the cross entropy method. The basic idea is to choose an initial set of values for the parameters and then iteratively improve those. The improvement is realized by minimizing the *cross entropy* between the chosen values and the theoretical optimal values. The cross entropy method was developed by Reuven Rubinstein. A tutorial, on which much of the following has been based, can be found in [3].

2.3.2 A more formal description

Basically, there are two things that can be important when using importance sampling. In many cases, the difference between the original distribution and the adjusted distribution is needed. It can be used to know how often a specific event would occur in the original model. In other cases, knowing the ratio is not as important as getting as close as possible to a choice of parameters that makes that specific event occur as often as possible. In that case, the goal is to get a distribution that always causes the event to happen. That is not very useful when one wants to calculate how often an event occurs in the real world, as simulation with the new distribution would basically give the same situation that has started the whole exercise: now, the 'normal' events have become rare. Thus, in some situations importance sampling can be used to get a distribution that lets specific events occur regularly, while knowing the 'amplification factor' between the original and adjusted distribution. In other situations the new distribution converges to a single result all the time and the relation to the original distribution is not important at all.

The latter case is true in optimization problems: the parameters used to initialize the algorithm are not very interesting, as long as the algorithm returns a solution that is close to or equal to the optimal solution. In the rest of this document, the cross entropy method is applied to optimization problems. Therefore, this description focusses on how to get close to an optimal solution and does not detail how the new probability distribution is related to the distribution of the original model.

The naive approach

Assume a model with random parameters $\mathbf{X} = (X_1, \ldots, X_n)$, with $0 \leq X_i \leq 1$ for $1 \leq i \leq n$. The values $\mathbf{X}$ assumes are based on the probability density functions $f(\cdot; \mathbf{v})$, where $\mathbf{v}$ represents the parameters of the distribution. Assume a function S that takes $\mathbf{X}$ and returns a real value. The goal is to maximize $S(\mathbf{X})$ by using a 'better' $\mathbf{X}$, which can be accomplished by adjusting $\mathbf{v}$.

The first step is to get an estimator for the chance that $S(\mathbf{X})$ is greater than or equal to some value γ. The real probability is

$$\ell = \mathbb{P}(S(\mathbf{X}) \geq \gamma) = \mathbb{E} I_{\{S(\mathbf{X}) \geq \gamma\}} \tag{2.1}$$

Here, $\mathbb{P}$ is a probability. I is the indicator function, which returns 1 if its argument is true and 0 if it is not. $\mathbb{E}$ is the expected value, in this case of I. As $I = 1 \iff (S(\mathbf{X}) \geq \gamma)$ and 0 otherwise, it is equal to the chance of $S(\mathbf{X})$ being equal to or larger than γ.

An easy estimator can be obtained by simple Monte-Carlo simulation:

$$\hat{\ell} = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\mathbf{X}_i) \geq \gamma\}} \tag{2.2}$$

where each $\mathbf{X}_i$ is a new sample of $\mathbf{X}$. That basically runs the simulation N times and averages the result, giving a better estimation as N becomes larger.

Smarter estimation: use importance sampling

So far, nothing special has happened: the described steps are just a normal simulation. If ℓ is expected to be a relatively rare event, a large number of simulations will have to be performed. The next step is thus to use importance sampling: take the values of $(X_1, \ldots, X_n)$ from a different distribution that improves the chance of $S(\mathbf{X}) \geq \gamma$. In the following equations, $\mathbf{u}$ is the original distribution and $\mathbf{v}$ is a new, supposedly improved distribution.

Monte-Carlo simulation using the new distribution should be more efficient, because the rare event will not be as rare anymore and thus fewer simulations are needed. In the simulation, one would correct for the 'amplification', giving a new estimator with the extra compensation factor:

$$\hat{\ell} = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\mathbf{X}_i \geq \gamma)\}} \frac{f(\mathbf{X}_i; \mathbf{u})}{g(\mathbf{X}_i)} \tag{2.3}$$

In this case, the samples are taken from the new distribution g and the compensation factor 'converts' between the old and new distribution[1].

Based on this definition of $\hat{\ell}$, the ideal distribution would be:

$$g^*(X) := \frac{I_{\{S(\mathbf{x}) \geq \gamma\}} f(\mathbf{x}; \mathbf{u})}{\ell} \tag{2.4}$$

because that would result in the following equality:

$$I_{\{S(\mathbf{X}_i) \geq \gamma\}} \frac{f(\mathbf{X}_i; \mathbf{u})}{g^*(\mathbf{X}_i)} = \ell \tag{2.5}$$

which gives the best estimator one could wish for:

$$\hat{\ell} = \ell \tag{2.6}$$

[1]This depends on g being non-zero where f is - a condition which is satisfied with the distribution that will be used in the end.

This estimator clearly has zero variance. There is one problem with this approach - g^* depends on ℓ, which is not known. Besides, it might be useful to have a g^* with certain characteristics, i.e. it might be useful if the distribution g^* can be represented by $f(\cdot; \mathbf{v})$. Depending on the choice of f, that might limit our ability to get the optimal answer, but we can still try to get as close as possible.

To get close to the optimal distribution, the difference between $f(\cdot; \mathbf{v})$ and g^* should be as small as possible. That raises the matter of how the difference between two distributions should be measured.

Intermezzo: difference between distributions

Consider two probability density functions g and h. We want to know how different h is from g, with the goal of having a h as close as possible to g. There are lots of possible definitions of 'different'. A few criteria for a good definition apply in this case.

Preferably, the difference should be zero if $h(x) = g(x) \forall x$. That ensures that it is easy to see that we've found the right solution: we surely can not do better than having identical results for all input values.

If $h(x) = 0$ in a region where $g(x)$ is not, the difference should be undefined or go to (minus) infinity. The reason is that in that case, the new distribution $h(x)$ clearly misses part of the support[2] of $g(x)$ and is thus not a good imitation of g. Note that it does not matter if $h(x) = 0$ at single values of x: the chance of selecting those individual values in a continuous distribution is zero, so the two distributions can for all practical purposes still be considered identical.

The requirement is a bit less strong the other way round: the difference can be defined, if $h(x) \neq 0$ for regions where $g(x) = 0$. In that case, the distributions are clearly not equal, so the difference should be non-zero.

Finally, it is important that the difference between $g(x)$ and $h(x)$ equals 0 only if both distributions are identical - thus, no two zero points should exist. If this condition does not hold, finding a difference of zero would still not tell us whether we've found the looked-for identical distribution. Another way to satisfy this criterium is to have a difference function which is convex and thus does not have multiple minima. In that case, finding a minimum guarantees that the difference between the distributions is minimal.

There are a lot of functions that meet those criteria. Another important one, which is not as easy to define, is that it should be something that is easy to calculate with - in the end, we should be able to adjust h based on how large the difference is and get closer and closer to g.

[2]The 'support' of a function is the set of points where its value is not equal to zero.

At this point, the name of the cross-entropy method becomes clear. The cross-entropy or Kullback-Leibler distance between $g(x)$ and $h(x)$ is defined as:

$$\mathbb{D}(g,h) = \mathbb{E}_g \ln \frac{g(x)}{h(x)} = \int g(x) \ln g(x) dx - \int g(x) \ln h(x) dx \tag{2.7}$$

Note that the Kullback-Leibler distance (KL-distance) is not really a distance. For instance, $\mathbb{D}(g,h) \neq \mathbb{D}(h,g)$, i.e. it is not symmetric.

The KL-distance satisfies the criteria defined before. If $h(x) = g(x)$, the distance becomes zero:

$$\ln \frac{g(x)}{g(x)} = \ln 1 = 0 \tag{2.8}$$

The second criterium states that distance should be undefined or go to infinity if $h(x) = 0 \neq g(x)$ $\exists \mathbf{x}$, where $\mathbf{x}$ is a region, not one or more individual points, as described on page 10. That holds as well for a single value of x:

$$\ln \frac{g(x)}{h(x)} = \ln \frac{g(x)}{0} = \text{undefined if } g(x) \neq 0 \tag{2.9}$$

It can be shown that this is the case if $h(x) = 0$ for some region j and $g(j) \neq 0$, because $\lim_{x \to j} g(x) \ln h(x) = -\infty$:

$$\begin{aligned}
\mathbb{D}(g,h) &= \int g(x) \ln g(x) dx - \int g(x) \ln h(x) dx && (2.10)\\
&= \int g(x) \ln g(x) dx - (\int_{x \subseteq \mathbf{x}} g(x) \ln h(x) dx && (2.11)\\
&\quad + \int_{x \nsubseteq \mathbf{x}} g(x) \ln h(x) dx) && (2.12)\\
&= \int g(x) \ln g(x) dx - \int_{x \neq j} g(x) \ln h(x) dx + \infty && (2.13)
\end{aligned}$$

The last equation only holds if $\mathbf{x}$ is a region of the points and not just a set of individual points, as discussed before.

Because the two integrals in the final equation will never go to $+\infty$ in our case, because we'll be using probability density functions, the final result will be undefined or go to infinity, as the criterium requires. Note that if $\mathbf{x}$ just contains individual points, the integral would not be over a region and thus not necessarily go to infinity.

The next looked-for property is that the function only has one zero point, which is reached when the distance between the two distributions is minimal. The KL-distance clearly satisfies that:

$$\ln \frac{g(x)}{h(x)} = 0 \iff \frac{g(x)}{h(x)} = 1 \Rightarrow g(x) = h(x) \tag{2.14}$$

It is shown on page 156 in [10] that the distance becomes zero if and only if $h(x) = g(x)$.

Thus, using the Kullback-Leibler distance or cross entropy as the measure of difference between the ideal and the generated probability density function is a good choice, provided it is easy to calculate with - which will be shown later on.

Next step: minimize the cross entropy

Given the difference between two distributions as defined above, the next step is to find a distribution $f(\mathbf{x}; \mathbf{v})$ that comes as close to g^* as possible. As can be seen from the definition of $\mathbb{D}(g, h)$, that comes down to maximizing $\int g(x) \ln h(x) dx$: that is the only part of the equation that contains $h(x)$, which is the only variable that can be adjusted.

Also, it is not possible to use any function $h(x)$. In the model, there might be limits to the values that the parameters can assume. Thus, $h(x)$ is replaced by $f(\mathbf{x}; \mathbf{v})$, where $\mathbf{v}$ is the vector with the parameters of the distribution. That results in the following maximization problem:

$$\max_{\mathbf{v}} \int g^*(\mathbf{x}) \ln f(\mathbf{x}; \mathbf{v}) d\mathbf{x} = \max_{\mathbf{v}} \int \frac{I_{\{S(\mathbf{x}) \geq \gamma\}} f(\mathbf{x}; \mathbf{u})}{\ell} \ln f(\mathbf{x}; \mathbf{v}) d\mathbf{x} \tag{2.15}$$

This can be written with stochastic variables and the expectation function as well:

$$\max_{\mathbf{v}} \mathbb{E}_{\mathbf{u}} I_{\{S(\mathbf{X}) \geq \gamma\}} \ln f(\mathbf{X}; \mathbf{v}) \tag{2.16}$$

Thus, we're looking for some distribution parameter $\mathbf{v}$ such that the equation above is maximal:

$$\mathbf{v}^* = \operatorname*{argmax}_{\mathbf{v}} \mathbb{E}_{\mathbf{u}} I_{\{S(\mathbf{X}) \geq \gamma\}} \ln f(\mathbf{X}; \mathbf{v}) \tag{2.17}$$

As before, Monte-Carlo simulation can be used to estimate the optimal value of $\mathbf{v}^*$. Using N independent samples $\mathbf{X}_i$, the estimator becomes:

$$\widehat{\mathbf{v}^*} = \underset{\mathbf{v}}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\mathbf{X}_i) \geq \gamma\}} \ln f(\mathbf{X}_i; \mathbf{v}) \tag{2.18}$$

The final step: split γ and parameter calculation

The original goal of the whole exercise described above was to find a maximum value for the function $S(\mathbf{X})$. One of the first steps was to define the following estimator:

$$\hat{\ell} = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\mathbf{X}_i) \geq \gamma\}} \tag{2.19}$$

If we want to find the highest possible value of $S(\mathbf{X})$, γ should be as high as possible - only by averaging over the very best solutions will the maximum value be found. We'll call the maximum value that $S(\mathbf{X})$ can reach γ^*.

Note that the difference between rare-event simulation and combinatorial optimization are very clear here. If we would be simulating a rare event, the value of $\hat{\ell}$ would be the measure of interest. In our case, however, we want the indicator function to return 1 all the time: which does not necessarily result in a good estimation of the chance that a certain event occurs, but it does give a very good idea of the maximum value that $S(\mathbf{X})$ can assume.

The values of $S(\mathbf{X})$ depend on the distribution used to generate $\mathbf{X}$. The parameters for that distribution are estimated with the following estimator, as has been discussed before:

$$\widehat{\mathbf{v}^*} = \underset{v}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\mathbf{X}) \geq \gamma\}} \ln f(\mathbf{X}_i; \mathbf{v}) \tag{2.20}$$

For any γ close to γ^*, a problem arrises with that estimator. The function $I_{S(\mathbf{X}) \geq \gamma}$ will almost never return 1 and thus the estimation of the parameters will not be very good. A lower γ, however, would give more 'hits' in the simulation, but at the same time cause problems with $\hat{\ell}$, which depends on $\gamma \approx \gamma^*$.

The solution to this is to adapt γ and $\mathbf{v}$ iteratively. If a good $\mathbf{v}$ is used, many 'hits' will occur even with a high γ. The high γ can in turn be used to generate a new estimate of $\mathbf{v}$. That results in the following algorithm:

1. Define an initial $\hat{v_0}$ and set a counter $t = 1$. The initial $\hat{v_0}$ can be a uniform distribution.

2. Generate samples $\mathbf{X}_1, \ldots, \mathbf{X}_N$ from the density function $f(\cdot; \mathbf{v}_{t-1})$ and set $\hat{\gamma}_t$ to such a value that only the best performing samples reach that value.

3. Use the same samples to calculate $\hat{\mathbf{v}}_t$.

4. Repeat from step 2 to until $\hat{\gamma}_t$ stabilizes at a maximum value.

Chapter 3

The Cross Entropy Ant System

The Cross Entropy Ant System (CEAS) is a routing protocol developed at NTNU that takes ideas from nature, specifically how foraging ants find their way, to build an efficient and robust routing protocol. A basic description can be found in [5]. It is an example of a swarm-based routing protocol. CEAS uses the cross entropy method to quickly converge to good routes.

This chapter gives a basic introduction to the cross entropy method and then describes how this method has been used in CEAS.

3.1 Foraging ants

As said, CEAS takes its basic idea from nature. Consider an area with an ant nest and a place with food, respectively called "s" and "d" for "source" and "destination". Each individual ant tries to find a way from the source to the destination. While travelling, it leaves a trail of pheromones. Those pheromones evaporate over time. Thus, if an ant uses a short route, it will pass there more often per unit of time on its way from the source to the destination and back again, resulting in a high concentration of pheromones on that route.

New ants choose their route partly based on the pheromones: the chance of taking a specific route is proportional to the amount of pheromones. The result is that the most efficient routes are used more often, which results in even more pheronomes on that path. Longer routes are chosen less often and eventually, the pheromone concentration gets lower and lower. The effect is clear: something which started as a random walk converges to an efficient path between source and destination. This phenomenon, where a group of systems do not communicate directly with each other, yet specific behaviour for the whole

group emerges, is called emergent behaviour.

3.1.1 Ants and rare events - the relation

So far, rare event simulation and foraging ants have been discussed. While the connection of foraging ants with routing is easy to see - both are about finding good routes to destinations - the connection between rare events and routing is less clear.

Generally, there are many paths to a certain destination. Only one or a few of those would be considered good paths. The problem in routing is to find those in the large collection of possible paths. Formulated that way, the connection with rare event simulation becomes apparent. In rare event simulation, an efficient path (i.e. series of steps) to a certain destination has to be found among a large number of possible paths.

3.2 CEAS

3.2.1 Introduction

A number of routing protocols, such as AntHocNet, are designed after the foraging ant behaviour described before. CEAS does that as well, but - contrary to the other protocols - bases it on the cross entropy method, which provides a mathematical basis for CEAS. The basic method is described in [8], though it is not directly applied to CEAS there.

The basic routing information in the protocol is a value $p_{t,r,s}$, which is the routing probability that a packet will go to node s, when it is at node r at iteration t. For all nodes, these probabilities can be grouped in a matrix $\mathbf{p}_t = p_{t,rs_{\forall rs}}$. The routing probabilities are the 'digital pheromones' of the protocol: the higher the pheromone value from r to s, the higher the probability that a packet will travel over that link as its next hop.

The protocol also uses a *temperature* γ_t, which converges to a value based on the cost of the best (lowest cost) route between two nodes. A performance function $h(\mathbf{p}, \gamma)$ indicates how good a certain matrix $\mathbf{p}$ is.

The algorithm works as follows:

1. Start with a set of routing probabilities $\mathbf{p}_{t=0}$ that are for instance uniformly distributed: all paths have the same probability $1/n$ with $n = size(s)$.
2. Generate a number of sample paths and select those samples that give

the best results - so find a γ as low as possible, that still gives a certain number of cases that satisfy $h(\mathbf{p}, \gamma)$.

3. Using those samples, generate a new matrix $\mathbf{p}_t$ which minimizes the distance (as defined before) between $\mathbf{p}_t$ and the optimal matrix. This optimal matrix is a matrix such that routes generated with it are the shortest routes, i.e. the matrix which, when used, results in the lowest possible temperature γ.

4. Increase t by 1 and repeat the procedure with the new matrix. Stop when the temperature γ stabilizes.

For $t \to \infty$, $\mathbf{p}_t$ gets an optimal solution, where $p_{t\to\infty,r,s}$ is either 1 (if the link between r and s is on the path with minimal cost) or 0 (if it is not). A more in-depth description can be found in section 2.3 in [6].

This procedure needs a number of samples before it can update the temperature and the matrix, which is not practical in a routing environment: it would be better if the probabilities would be updated with each arriving routing packet, instead of waiting for a batch of packets to arrive. CEAS achieves that by adjusting the performance function every time new information arrives, basing it on both the new information and the currently used function.

3.2.2 Implementation

To implement the behaviour described above, the routing protocol uses packets called *ants*. Ants travel from a source to a destination, possible using the matrix $\mathbf{p}_t$ to choose the route, accumulating the cost of the followed path. At the destination, the temperature is updated. The ant then travels back along the same way to the source, updating $\mathbf{p}_t$ in the nodes it passes.

3.2.3 Avoiding converging to a local optimum

With the algorithm described above, there is a risk of ending up with a path that is not actually the shortest. At first, a reasonable - but not optimal - path p_R may be found. No other paths have any pheromone at all, so all ants choose to follow p_R. No better path is ever found.

To avoid that situation, CEAS knows two different kind of ants. *Normal forward ants* use the pheromone tables to find their destination, while *explorer ants* simply pick a random next hop, not using the pheromone values at all. Using a suitable mix of normal ants and explorer ants results in new routes being found, while known routes get the right amount of pheromone.

That does not solve the whole problem, however. Consider the situation where the reasonable path p_R is known. At some point, a shorter route p_B is found.

However, the chance of taking that route is not very high, as only one (explorer) ant 'deposited' pheromone on that route, compared to possibly thousands of ants on p_R. Thus, p_B is not chosen more often and its pheromone value is not reinforced.

To avoid this situation, a basic solution is to sample a number of random paths at first, before starting to use the pheromone values. Thus, the default implementation of CEAS has an *initialization phase* during which only explorer ants are sent. The pheromone tables then contain reasonable levels, after which the normal system with both normal and explorer ants can take over to maintain the short routes and adapt to changes in the system.

3.2.4 Extensions

The system described above works, but it is not as efficient as it could be. A number of improvements are already known that speed up convergence or reduce the overhead of the protocol by limiting the number of ant transmissions needed to get good routes.

Elite selection is one such optimization. When a forward ant reaches its destination, it is only converted to a backward ant if the cost for the path it has followed is within a certain distance from the best path known to that destination. Other ants are discarded. In this way, ants that would only confirm that a low-chance path is indeed not worth taking are not propagated, thus causing a reduction in overhead. Elite selection also helps to avoid the problem of converging to a local optimum, described before. A more thorough description of elite selection, including benchmarks, can be found in [7].

Another optimization can be found in the rate ants are generated. When a network is changing, a high number of ants is needed to find the best (new) routes. However, if no changes occur, it would be better to create less ants. To solve this problem, ant rates are self-tuning. If almost no forward ants return to the source node, that node knows that the protocol has likely not converged to a stable state yet - ants either do not get to the destination at all or if they get there, they are not converted to backward ants because they do not qualify as elite ants. In a stable network, most ants reach the destination over the lowest-cost path and are thus within the elite selection range. That implies that the number of ants that is generated can be lowered when the number of forward and backward ants is almost equal.

Chapter 4

Performance conditions in dynamic networks

In this chapter, various conditions will be discussed that influence the performance in dynamic networks. Knowledge of the conditions will be useful in the next chapter, which describes various routing protocols and gives an idea of their behaviour when said conditions change. Because all protocols under discussion are routing protocols, this chapter focuses on those conditions that directly influence routing.

4.1 Performance

To define the performance conditions, an important first step is to define 'performance' in the context of (routing in) dynamic networks. I assume a network with the following characteristics to keep this definition as general as possible:

- Two or more nodes
- Zero or more connections between nodes
- Zero more data flows that want to travel from one node to one or more - not necessarily connected - other nodes. Data flows consist of packets.

In such a network, an important performance metric is whether packets arrive at all: if a route exists between source and destination of a packet, the packet should preferably get to its destination. Thus, the *delivery ratio* is a performance indicator. Also, it might be important to notify the sender when a packet could not be delivered.

Not only does it matter whether a packet arrives at all, in many cases it is also nice if it does not take much time to get to its destination. *End-to-end delay* is therefore another indicator. For routing protocols, this translates in a few more specific performance indicators. In many cases, not only the average speed is important, but also the speed variation or jitter, which is especially important for real-time communication between nodes: in that case, a low *average* delay might not be very useful if that is achieved by sometimes having a very low delay and at other times having a high delay. The peaks of high delays degrade the quality of for instance voice chats, which depend on not having sudden long delays. Many papers include delay measurements - it is relatively easy to measure.

Especially in dynamic networks, it is to be expected that nodes appear and disappear regularly. Also, connections between nodes change: new connections form and existing connections break or degrade. *Adaptibility to topology changes* is important for a routing protocol.

Scalability is another important characteristic of networks. Routing schemes that work in a relatively small network might fall apart when the number of nodes or connections grows: for instance, if the number of control messages a protocol needs grows faster than linearly with the number of nodes in a network, the number of messages *per node* will increase and at some point, each node spends all of its time on routing and none on actually processing any data. Of course, such a linear increase is probably impossible to reach, but to make a protocol usable in larger networks, overhead should not increase too fast.

A routing protocol should work well in many different circumstances. Not only should scale from small to large networks, it should preferably also work in both dynamic and static situation, networks with high and low bandwidth, densely connected or sparse. The ability to work in many different environments without much user intervention could be called the *flexibility* of a protocol.

The relative importance of each performance measure differs between situations - in some networks, jitter might not be important at all and in other networks, scalability problems will never occur.

Most research seems to be on maximizing the delivery ratio in mobile networks, as well as minimizing the overhead and increasing the scalability of networks. However, no agreed-on environment seems to exist: all papers use different network simulators, different network sizes, different node movement behaviours and different traffic models. That makes it difficult to compare the protocols, even if they are basically designed for the same task. In my thesis, choosing an environment in which at least a number of routing protocols can be compared will be important. Some more information on performance characteristics in mobile networks can be found in [12].

Chapter 5

Routing protocols in dynamic networks

This chapter describes some protocols already developed for use in dynamic networks. A few categories of routing protocols can be defined:

- Pro-active routing
- Reactive routing
- Hybrid (both pro-active and reactive) routing

One pro-active and one reactive routing protocol are described, as well as two hybrid protocols.

To avoid having to introduce new variables all the time, a few standard names for nodes are used in this chapter. In the next sections, A is always the source node for a packet or flow, B is the destination and any intermediate nodes are denoted by I.

After each protocol description, a section is included in which the (expected) performance of the protocol on the metrics defined in the last chapter is discussed:

- Delivery ratio
- End-to-end delay and jitter
- Adaptability to topology changes
- Scalability and flexibility

The last section contains a selection of benchmark papers in which the discussed protocols are simulated and compared to other routing protocols.

5.1 Pro-active routing: Destination-Sequenced Distance-Vector Routing (DSDV)

5.1.1 Introduction

In pro-active routing, routes to destinations are kept up-to-date regardless of whether there is actual traffic to a certain destination. An example of such a pro-active routing protocol is Destination-Sequenced Distance-Vector Routing or DSDV. It was proposed by Perkins and Bhagwat in 1994 in [14] and is based on the Bellman-Ford algorithm, with changes to make it usable in dynamic networks.

In general, an advantage of pro-active routing is that packets can be delivered relatively fast: because routes are kept up-to-date all the time, packets can be transmitted immediately when they arrive. A disadvantage naturally follows: if there is not much traffic or routes change very frequently, the overhead of keeping the routing tables up-to-date will be large.

5.1.2 The protocol

Each node has a table with the following information about each known destination:

1. Hop count to destination
2. Next node to destination
3. Sequence number of last routing update for this destination

The hop count may be replaced by a different cost function, such as the time required to reach the destination.

Packets travel from A to intermediate node I_1 as indicated in the routing table. At I_1, it is sent to I_2, which is the next node recorded in the routing table at I_1 to get to B. This goes on until at last the packet arrives at B.

To maintain consistent routing tables, all nodes transmit updates regularly and always when a significant routing change occurs. There are two types of updates: *full updates* contain information about all known destinations, while *incremental updates* only contain part of the routing table. For each destination in an

update, the cost to that destination is included, as well as the last sequence number known from that destination. In addition to that, an update contains a new sequence number for the hosts which send the update. All receivers look at the sequence number and in case it is larger than the latest sequence number they have received so far from that host, perform updates as necessary for each received destination:

- If the sequence number for that destination is smaller than the sequence number already stored, nothing happens.
- If the sequence number is larger than the currently stored number, the cost to that destination replaces the currently stored cost.
- If the sequence number is equal to the stored number, the smaller of the currently stored cost and the new cost is stored in the routing table.

If any changes to the routing table are made as a result of processing the received updates, the node sends an update itself[1]. Incremental update packets always contain all changes relative to the last transmitted full update. That implies that incremental updates grow over time, as more and more changes to the routing table occur. To minimize the traffic generated by routing updates, nodes can decide to send a full update regularly. That will cause the following incremental updates to be smaller and thus, if the full updates are sent at the right moment, to lower the amount of traffic needed for routing updates. The protocol does not define when full updates should be sent.

To minimize the trafic even more, not all changes made to a routing table at a node are immediately forwarded to the other nodes: only significant changes are propagated. A mechanism is proposed where the significance of an update is related to the rate of updates for that destination received so far. If many changes have happened, new updates become less significant and are not broadcasted. This mechanism prevents a storm of updates when the cost function to a destination is still stabilizing. However, even if changes are not forwarded to other nodes, a node always uses the latest information it has received. That implies that a 'public' routing table exists that might differ from the internally used routing table.

If a node I_2 detects that its connection to destination B is lost, it has to forward that information to its neighbours. A potential problem arrises, because B can not provide a sequence number for this update. If I_2 makes up a sequence number on its own, it might clash later on with the real sequence numbers generated by B. To avoid that problem, an odd number is set as the sequence number and the cost to get to B is set to ∞, thus causing an update to be

[1]The description by the original authors of the protocol leaves room for multiple interpretations of this rule. It could be read as only sending an update (and generating a new sequence number) if *anything* in the routing table changes or only if a metric in the table is changed. In the latter case, no update is transmitted when a new sequence number is received that does not change any metric. The option of always sending an update is shown to give the best results.

sent. Normal sequence numbers are always even, so any new sequence number generated by B will immediately replace the number generated by I_2.

5.1.3 Performance characteristics

DSDV was one of the first protocols to be proposed for use in mobile networks. Because it keeps an up-to-date view of the network at all times, setting up a connection should be fast. As the protocol does not specify how topology changes should be noticed - it assumes that a lower layer will notify the routing protocol when the link cost changes or a node appears or disappears - there is no easy method to say whether the protocol is able to work with topology changes very well. Given that such changes are noticed, the protocol will include them automatically - although settling times can be high. High settling times are mostly caused by not immediately forwarding a new routing table when the cost of a link changes - which lowers the overhead and avoids 'update storms', but that comes with the cost of routing tables that are outdated for a longer time.

Reliability of the network may become a problem, because DSDV does not keep backup paths.

Scalability is an issue, because nodes keep routes to all other nodes. Thus, a change in the cost of a single link can cause a large number of updates to be broadcasted in large parts of the network. Each update can be fairly large, especially if updates are frequent. In that case, even incremental updates can have a significant size. This effect is limited to a certain extent by only forwarding significant routing updates.

5.2 Reactive routing: Ad hoc On-Demand Distance Vector (AODV)

5.2.1 Introduction

One of the original authors of DSDV has later on proposed an algorithm that uses a very different approach to routing in dynamic networks [2]. In Ad hoc On-Demand Distance Vector (AODV), no routes are calculated until actual data has to travel from A to B. The protocol is documented in an experimental RFC [13].

5.2.2 The protocol

When a new route is needed, a route request (RREQ) packet is transmitted to all neighbours. It contains the following information:

Source address The address of the node that produced the RREQ.

Source sequence number The current 'version' of the information of this node.

Broadcast ID A number that is increased with each RREQ that a source produces.

Destination address The address of the destination.

Destination sequence number The minimum sequence number of the information about the destination, i.e. how old the information about the route to the destination is allowed to be.

Hop count The number of hops the RREQ has taken so far.

The combination of source address and broadcast ID is unique. Each node that receives a RREQ either forwards it if it does not know how to reach the destination, increasing the hop count, or returns a route reply (RREP) packet if it knows how to reach the destination. Because forwarding in this case implies broadcasting, a node may receive multiple copies of a RREQ. Such copies are dropped.

If a node forwards a RREQ, it locally stores some information about the request:

- Destination address
- Source address
- Broadcast ID
- Expiration time for reverse path route entry
- Sequence number of source node

At some point, the RREQ will reach a node that either *is* the destination or knows how to reach it. In the latter case, it checks whether its own sequence number for that destination is newer than the sequence number the RREQ asks for. If it is, a route reply (RREP) is constructed. The RREP travels back along the path that the RREQ followed, using the information stored at each intermediate node about the request. At some point, it will reach the source node and both the source node and all intermediate nodes have a route to the destination.

It is possible that an intermediate node receives multiple RREPs for the same RREQ. In that case, it forwards RREPs after the first one only if its destination sequence number is larger than the RREPs forwarded before (indicating a newer path) or if the destination sequence number is the same, but the hop count is lower (indicating a better path). Source nodes can start to transmit data as soon as they receive a RREP - its routing table can be updated if newer RREPs arrive and point to better paths.

It is also possible that an intermediate node which forwarded a RREQ does not receive any RREPs at all. In that case it is not on the path between source and destination and after the expiration time, which was stored when the RREQ arrived, the routing information is deleted.

Nodes update their routing table according to a few simple rules. When a RREP arrives, it is always used to overwrite the current routing entry for that destination if its destination sequence number is newer than the currently stored one: the RREP has newer information. If the sequence number is equal to the current sequence number, it only replaces the current entry if the hop count is smaller than the currently known best hop count. That way, nodes always use the best up-to-date routing information possible.

When a node notices that a link to one of its destinations breaks, for instance because one of its neighbours disappears, it produces a special RREP packet. The destination sequence number in that packet is one higher than the locally stored number for that destination, thus triggering updates at all receivers that use this node to send data to that destination. The cost of the path is set to ∞. Any node that wants to create a new connection to the destination has to send a RREQ with an updated destination sequence number, forcing a discovery process for other possible routes to the destination.

5.2.3 Performance characteristics

Because of its reactive nature, path setup times are higher than with DSDV. However, that helps with scalability: only if a node is actively involved in data transmission, does it have to store routing information - and even then, only of those nodes that it has to interact with. AODV does not have many parameters that have to be optimized by hand - it should not be much work to deploy it.

Various researchers have performed benchmarks to compare DSDV and AODV. Sesay and Johansson show that in almost all cases, AODV is the preferable routing protocol. DSDV has a higher overhead because it tries to keep an updated view of the network at all times. That makes AODV more scalable and gives AODV higher throughput as well, because the reduction in overhead can be used for the transmission of actual data.

5.3 Hybrid routing (1): AntHocNet

5.3.1 Introduction

AntHocNet, developed by Gianni Di Caro and others [1], is based on the same ant behaviour used in the design of CEAS. However, it uses reinforcement learning to converge to the shortest paths. It is shown to be more efficient than AODV in at least some situations. For an introduction to how ants leave pheromones to converge to a short path between their nest and a food source, see the first part of this document.

5.3.2 The protocol

AntHocNet is based on routing packets that travel through the network that are called 'ants'. Part of the protocol is reactive: it only builds a route when actual data has to travel to a certain destination. However, it uses a proactive approach by maintaining the path and looking for better routes to that particular destination once a path has been found.

Nodes periodically broadcast 'hello messages' to inform their neighbours about their presence. If no such message has been received from a node from which such a message has been received earlier, the connection to that node is considered broken. Each node has a pheromone table, which indicated the 'goodness' of paths to different destinations. When ants travel through a node, they update the pheromone table.

To find a route to a certain destination, a node broadcasts a 'forward ant'. At each node the ant visits, two things can happen:

- The forward ant chooses a next hop based on the routing information at that node. 'Good' paths have a higher chance of being chosen.
- The forward ant is broadcasted to all neighbours again.

To limit the number of ants, two mechanisms are used. Each ant has a maximum number of hops it is allowed to travel. If it does not reach its destination in at most that many hops, the ant is discarded. A second, more important rule is that not all ants are forwarded by all nodes. If a node I has already received an ant from the same source on the way to the same destination, it can decide to drop it - another ant has clearly already found a faster way to reach I, so there is not much point in forwarding it anymore. Ants are only dropped if the route they have found is worse by at least a certain factor, so it is still possible that more than one ant is forwarded at I. Also, the first step that the ant took is considered. If an ant is worse than another ant that has passed before, but has a different first hop than the better ant, it will still be forwarded. This avoids

converging to only one route for the first number of hops, which would limit the number of good paths found - which in turn leads to less redundancy and reliability.

Once an ant reaches its destination B, it is converted to a 'backward ant'. It travels back along its route and updates the routing table at each node with information about the cost to get to B.

The new 'routing chance value' at a node I is calculated as follows:

$$\tau_B^I = (\frac{\hat{T}_B^I + h * T_{hop}}{2})^{-1}$$

Where

- $\hat{T}_B^I$ is the estimated time to get from I to B (as recorded by the ant)
- h is the number of hops to B
- T_{hop} is the time one hop takes in unloaded network conditions (hardcoded)

The estimation thus averages the calculated travel time and the unloaded travel time and inverses this, giving lower 'routing chances' when travel times are longer. This value is then combined with the existing values to update the value in the routing table as follows:

$$\mathcal{T}_{NB}^I = \gamma \mathcal{T}_{NB}^I + (1 - \gamma)\tau_B^I \qquad \gamma \in [0, 1]$$

Here, $\mathcal{T}_{NB}^I$ is the chance that a packet at I will choose N as the next hop to B - so N is the last node the backward ant has visited before it arrived at I. The parameter γ influences the weight of the new information in comparison to the current value.

Once a path has been found, data can travel from A to B. This is always done stochastically, where the best routes have a higher chance of being chosen.

The stochastic mechanism is implemented in an easy way. As shown above, the routing table does not really store cost values to destinations, but stores 'chances' P_{NB} of the next hop N being used to travel to destination B. When a forward ant has to choose how it will travel further, the probability for each possible next hop is calculated:

$$P_{NB} = \frac{(T_{NB}^I)^\beta}{\sum_{j \in \mathcal{N}_B^I} (T_{JB}^I)^\beta} \qquad \beta \geq 1$$

where $\mathcal{N}_B^I$ is the set of neighbours that can be used to reach B. The parameter β can be used to lower the chance of very explorative behaviour. For data packets, a higher value of β is used, causing them to use the best-regarded paths, while forward ants use a lower value and thus more often choose routes which are less likely to be good.

During a data session, new forward ants are created regularly. They obey to the same rules as normal forward ants: most of the time it will use one of the existing routes, but sometimes it is broadcast to all neighbours of a node. This explorative behaviour helps to find new, better routes.

When a link or node fails, its neigbours will not receive hello messages any longer. It will consider the path to be broken and generate a 'route repair ant', which basically has the same task as a forward ant: to find a route to the destination. Once it finds a path, the routing table will be updated and the new path will be used instead.

5.3.3 Performance characteristics

The designers of AntHocNet have run a number of benchmarks against AODV. Under certain conditions, AntHocNet is shown to have a higher delivery ratio and less end-to-end delay than AODV. That comes at a cost, however: overhead of AntHocNet can be several times higher than AODV.

AntHocNet should scale relatively well: routing information is generated only for nodes that are actually sending data to each other. There is a risk that finding a path to a new destination floods the network with a lot of ants. The AntHocNet authors suggest to mitigate this effect by using the hello messages to spread basic routing information. That way, forward ants have a better idea of where to go and should thus reach their destination more efficiently.

In benchmarks by Ducatelle [4] and Kudelski [11], AntHocNet has a higher packet delivery ratio than AODV. Its end-to-end delay is in most cases lower as well. These results hold both at low and high node mobility. Only at high data rates, when the network gets congested, AODV performs slightly better, according to Ducatelle: it sends less control packets and each packet is smaller.

One concern with AntHocNet is the number of parameters that currently have to be hardcoded in the protocol. There are exploration factors for forward ants and data, there is the maximum hop count which depends on the network diameter, there are different smoothing factors and an acceptance parameter - and the optimal values of all of them depend on the characteristics of the network. While the benchmarks show a good results, it is possible that configuring a network for such performance requires manually optimizing the parameters, which reduces the flexibility of the protocol.

5.4 Hybrid routing (2): Hazy Sighted Link State (HSLS)

5.4.1 Introduction

The Hazy-Sighted Link State routing protocol (HSLS) is a routing protocol that was proposed by Cesar Santiváñez and Ram Ramanathan from BBN Technologies in 2001 [15]. In their paper, they define the goal of routing protocols to be the minimization of overhead. Overhead is defined as 'the total amount of bandwidth used in excess of the minimum amount of bandwidth required to forward packets over the shortest distance by assuming that the nodes had instantaneous full-topology information'. With this definition, inefficient routing (i.e. taking longer routes than strictly necessary) contributes to the overhead. The goal of a routing protocol should be to minimize the overhead in any type of network.

According to the designers, pure pro-active protocols have to transmit (in the worst case) a message every time a link change is detected. Each message has to be retransmitted by each node in the network to reach all nodes. The HSLS authors argue that, because both the number of link changes and the number of transmissions needed increases linearly with the the network size N, the total overhead of a purely pro-active protocol grows with N^2.

A similar argument is given for pure reactive protocols. A new protocol message has to be transmitted for each new connection and then has to be retransmitted by each node in the network. Because the number of connections and the number of retransmissions per connection grow linearly with N, the total overhead of purely reactive protols grows with N^2.

Of course, it is possible to avoid the worst cases, for instance by storing routing information temporarily in a reactive protocol. In that case, the protocol is not purely reactive anymore - but it will likely be more efficient in terms of overhead.

The designers argue that the theoretical optimum is somewhere in between: a protocol should be hybrid. The set of Fuzzy Sighted Link State (FSLS) algorithms is proposed. FSLS algorithms are based on the link state (pro-active) approach, but limit the overhead in both the space and time dimension. To limit the spatial overhead, not all link changes are propagated through the complete network. By not transmitting updates when they occur, but collecting them before they are transmitted, the overhead in the time dimension is reduced.

Of course, not transmitting all updates through the complete network immediately causes inefficient routes to exist for longer than necessary. That implies that a new type of overhead is created: the 'sub-optimal routing overhead'. An optimal routing protocol, i.e. with minimal overhead, should balance the overhead caused by control messages and the sub-optimal routing overhead to

minimze the total overhead. That protocol is the Hazy Sighted Link State routing algorithm.

5.4.2 The protocol

HSLS is a link state algorithm that does not always transmit its updates to all nodes in a network at all times. The design seeks a balance between not causing control message overhead and limiting inefficient routing overhead.

Limiting the overhead in time is done by only sending updates every t seconds, aggregating all link changes that have occured in the last t seconds in a single control message.

Limiting the overhead in space is done by limiting the range that a control message will travel. Some messages will just reach the direct neighbours of a node, while other messages will propagate through the entire network.

After a number of calculations, the authors of HSLS propose a protocol that sends control messages (called Link Status Updates or LSUs) with range r every $2^{r-1} * t$ seconds. Thus, a node transmits a message to its neighbours every t seconds. Every $2 * t$ seconds, a message is sent to all nodes 2 hops away. Of course that LSU passes each neighbour as well, so no separate LSU (which would have the same information) is sent to the neighbours. That results in the following set of LSUs being sent:

Time (t seconds)	**Range (hops)**
1	1
2	2
3	1
4	3
5	1
6	2
7	1
8	4
9	1
10	2

When the range r reaches the (best known) distance to the furthest node, t (and thus r) is reset. The protocol has an initialization phase in which LSUs with an unlimited range are sent. That way, the distance to the furthest node is known.

Using this protocol, closer nodes have a more up-to-date view of the state of a node than other nodes that are further away.

5.4.3 Performance characteristics

The authors of HSLS compare it to a number of other FSLS routing protocols, which have different algorithms to limit the transmission of control messages in space and time. For instance, in Discretized Link State (DLS), messages are still sent every t seconds, but the range is always ∞, so effectively no limitation of space is used. Near Sighted Link State is another example: updates are transmitted every t seconds with range k, unless t is a multiple of some value v - in that case, the range is set to ∞.

HSLS performs better for large networks than the other protocols that are tested: the throughput (defined as the percentage of packets that arrive at the destination) is highest of all.

5.5 Benchmarks

In a number of papers, simulations have been run to compare different routing protocols. The next subsections briefly detail the most important conclusions of some of these papers.

5.5.1 Sesay

In [16], DSDV, AODV, TORA and DSR are compared on throughput, delay, overhead and route acquisition time. DSR (Dynamic Source Routing) is a reactive routing protocol somewhat comparable to AODV, the main difference being it uses source routing: the complete path of a packet is attached to the packet at the source. Intermediate nodes don't decide on the next hop, but simply follow the 'instructions' attached to the packet.

The article concludes that AODV is more scalable than DSDV, has less overhead and has higher throughput. However, it takes more time to find a route (because of its reactive nature) and end-to-end delay is worse in small networks as well. TORA, a protocol which supports multiple routes and multicasting, is usable in higly dynamic networks. The best all-round performer is AODV.

5.5.2 Johansson

In [9], DSDV, AODV and DSR are compared in scenarios where nodes either move randomly or with predefined speeds and directions. It is shown that DSDV does not work very well in dynamic networks: a reactive approach is more efficient in such networks. DSR and AODV are both good candidates to use in the simulated scenarios, with DSR performing better in smaller networks

with less traffic and AODV showing better results in more higly utilized, bigger networks.

5.5.3 Kudelski

In a recent comparison by Kudelski and Pacut in [11], AODV, AntHocNet and their own protocol, AntHocGeo, are compared. The simulations focus on the end-to-end delay and the delivery ratio. In a simulation with moving nodes, AntHocNet has a lower end-to-end delay and a higher delivery ratio than AODV, with different node speeds. It is not stated how the nodes choose their route - most likely, a random waypoint model has been used.

5.5.4 Ducatelle

Ducatelle, Di Caro and Gambardella, the designers of AntHocNet, have compared their protocol with AODV regarding the end-to-end-delay and the delivery ratio [4]. Under high load, the end-to-end delay of AntHocNet is larger than AODV because the control packets are larger in size and number and thus interfere with the data packets. Just like the results reported in [11], AntHocNet has a higher packet delivery ratio and in most cases a lower delay than AODV.

Chapter 6

Conclusion

As stated in the introduction, this research project is a preparation for my master thesis at NTNU. Two goals were defined: describing the Cross Entropy Ant System (CEAS) and giving an overview of routing in dynamic networks.

The basic algorithms of CEAS have been described in this report, including existing improvements to the basic protocol. What has not been described are implementation details like the format of the routing packets - that will be done during my thesis work. Not in the report, but studied nevertheless, are the benchmarks which show how much of a difference specific improvements to CEAS actually make.

To get an overview of routing in dynamic networks, four protocols were selected, which were as different from each other as possible to get a broad idea of the possible approaches to routing in such an environment. DSDV and AODV are relatively basic protocols, while both hybrid systems are more advanced. AntHocNet shares the idea of imitating ants to perform routing with CEAS, but it takes a less mathematical approach. HSLS tries to provide a rigorous mathematical basis to its design which is interesting, but does not seem to have gained much attention by other researchers. Conclusions from a number of simulation papers are included in the report, which compare the performance of different combinations of DSDV, AODV and AntHocNet. No papers with benchmarks of HSLS were found, so a comparison with that protocol could not be made.

One field that was not covered in this report, but will be important when writing my thesis, is what simulation environment should be used. The papers that describe the four protocols use four different network simulators. Parameters like network density, mobility of nodes and type and intensity of traffic flows are not standardized either, so it is difficult to compare the existing protocols. Choosing a simulation environment should be part of the further work for my thesis.

Bibliography

[1] Gianni Di Caro, Frederick Ducatelle, and Luca Maria Gambardella. AntHocNet: An Adaptive Nature-Inspired Algorithm for Routing in Mobile Ad Hoc Networks. Technical report, 2004.

[2] Charles E. Perkins. Ad-hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[3] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, 134:19–67, 2005.

[4] Frederick Ducatelle, Gianni Di Caro, and Luca Maria Gambardella. A study on the use of MANETs in an urban environment. Technical report, 2007.

[5] Poul E. Heegaard, Bjarne E. Helvik, and Otto J. Wittner. The Cross Entropy Ant System for Network Path Management. *Teletronikk*, 2008.

[6] Poul E. Heegaard, Otto Wittner, Victor F. Nicola, and Bjarne Helvik. Distributed Asynchronous Algorithm for Cross-Entropy-Based Combinatorial Optimization.

[7] Poul E. Heegaard, Otto Wittner, Victor F. Nicola, and Bjarne Helvik. Distributed Asynchronous Algorithm for Cross-Entropy-Based Combinatorial Optimization.

[8] Bjarne E. Helvik and Otto J. Wittner. Using the Cross-Entropy Method to Guide/Govern Mobile Agents Path Finding in Networks. In *Mobile Agents for Telecommunication Applications*, pages 255–268. 2001.

[9] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, and Mikael Degermark. Scenario-based Performance Analysis of Routing Protocols for Mobile Ad-hoc Networks. 1999.

[10] J. N. Kapur and H. K Kesavan. *Entropy Optimization Principles with Applications*. Academic Press, Inc., 1992.

[11] Michal Kudelski and Andrzej Pacut. Ant Routing with Distributed Geographical Localization of Knowledge in Ad-Hoc Networks. 2009.

[12] Sampo Naski. Performance of Ad Hoc Routing Protocols: Characteristics and Comparison, 2004.

[13] C. Perkins and E. Belding-Royer. RFC 3561: Ad Hoc On-Demand Distance Vector (AODV) Routing, 2003.

[14] C. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. 1994.

[15] Cesar Santiváñez and Ram Ramanathan. Hazy Sighted Link State (HSLS) Routing: A Scalable Link State Algorithm. Technical report, 2003.

[16] Samba Sesay, Zongkai Yang, Biao Qi, and Jianhua He. Simulation Comparison of Four Wireless Ad hoc Routing Protocols. *Information Technology Journal*, pages 219–226, 2004.

Bibliography

[1] The network simulator - ns-2. [Online]. Available: http://www.isi.edu/nsnam/ns

[2] S. Kurkowski, T. Camp, and M. Colagrosso, "Manet simulation studies: the incredibles," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 4, pp. 50–61, 2005.

[3] P. E. Heegaard, O. Wittner, V. F. Nicola, and B. Helvik, "Distributed Asynchronous Algorithm for Cross-Entropy-Based Combinatorial Optimization," ca. 2005.

[4] V. Kjeldsen, O. J. Wittner, and P. E. Heegaard, "Distributed and scalable path management by a system of cooperating ants," in *Proc. International Conference on Communications in Computing (CIC)*, Las Vegas, NV, USA, Jul. 2008.

[5] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artifical Systems*. Oxford University Press, 1999.

[6] G. Di Caro and M. Dorigo, "AntNet: Distributed stigmergetic control for communications networks," *Journal of Artificial Intelligence Research (JAIR)*, vol. 9, pp. 317–365, 1998.

[7] M. Dorigo, G. Di Caro, and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artificial Life*, vol. 5, no. 2, pp. 137–172, 1999.

[8] B. E. Helvik and O. J. Wittner, "Using the cross entropy method to guide/govern mobile agent's path finding in networks," in *Proc. International Workshop on Mobile Agents for Telecommunication Applications (MATA)*, Montreal, Canada, Aug. 2001.

[9] P.-T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A Tutorial on the Cross-Entropy Method," *Annals of Operations Research*, vol. 134, pp. 19–67, 2005. [Online]. Available: http://www.springerlink.com/content/kpw596202975755n

[10] P. E. Heegaard and O. J. Wittner, "Overhead reduction in distributed path management system," *Computer Networks*, vol. 54, no. 6, pp. 1019–1041, Apr. 2010.

[11] P. E. Heegaard, B. E. Helvik, and O. J. Wittner, "The Cross Entropy Ant System for Network Path Management," *Teletronikk*, 2008.

[12] B. E. Helvik and O. J. Wittner, "Using the Cross-Entropy Method to Guide/Govern Mobile Agents Path Finding in Networks," in *Mobile Agents for Telecommunication Applications*, 2001, pp. 255–268. [Online]. Available: http://www.springerlink.com/content/8q470wpybc8266ur

[13] V. Kjeldsen, "Coperation through pheromone sharing in swarm routing," 2007.

[14] *ns-3 project description*. [Online]. Available: http://www.nsnam.org/docs/proposal/project.pdf

[15] *ns-3 reference manual*. [Online]. Available: http://www.nsnam.org/docs/release/manual.pdf

[16] P. E. Heegaard and I. Fuglem, "Demonstrator 1: Ant-based monitoring on software IP routers," BISON (IST-2001-38923), Tech. Rep., 2006. [Online]. Available: http://www.cs.unibo.it/bison/deliverables/D14.pdf

[17] I. S. I. U. of Southern California, "RFC 791: Internet Protocol v4," 1981. [Online]. Available: http://www.ietf.org/rfc/rfc791.txt

[18] Waf. [Online]. Available: http://code.google.com/p/waf

[19] *The* ns *Manual*, Kevin Fall and Kannan Varadhan ed., The VINT Project, UC Berkeley, LBL, USC/ISI, and Xerox PARC.

[20] F. J. Ros and P. M. Ruiz, "Implementing a new manet unicast routing protocol in ns2," Dec. 2004. [Online]. Available: http://masimum.inf.um.es/nsrt-howto/pdf/nsrt-howto.pdf

[21] L. Paquereau and B. E. Helvik, "Simulation of wireless multi-* networks in ns-2," in *Proc. Workshop on NS2 (WNS2)*, Athens, Greece, Oct. 2008.

[22] NSF Grant "Frameworks for ns-3". [Online]. Available: http://nsf.gov/awardsearch/showAward.do?AwardNumber=0958015

[23] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley, "ns-3 project goals," in *Proc. First Workshop on NS2 (WNS2)*, Pisa, Italy, Oct. 2006.

[24] Google Summer of Code. [Online]. Available: http://code.google.com/soc

[25] Trådløse Trondheim. [Online]. Available: http://www.tradlosetrondheim.no