Extensions on Contract Net Protocol for AGVs

Multi-agent systems

Jens Claes Victor Le Pochat

June 3, 2016

1 Introduction

1.1 Objectives

The objective of this report is to investigate whether two extensions of Contract Net (CNET) [2, 8], namely Contract Net with Confirmation Protocol (CNCP) [6] and Dynamic Contract Net (DynCNET) [9] are an improvement on CNET itself.

We do this in a drone delivery setting where a company owns a fleet of drones (or UAVs, unmanned aerial vehicles) that receives orders from clients, collects the ordered package at one of several warehouses and delivers them to that client. The drones autonomously receive, process and deliver the client orders. The goal is to maximize the company's profit, by having the most suitable drone serve each order. Clients have a time limit for their deliveries, after which the company needs to pay a fine. Drones can break down and are constrained by their batteries, which can be charged at a warehouse. Drones cannot collide with each other.

We investigate whether any of the aforementioned protocols are suited for a drone company to be profitable in the described pickup and delivery problem. We will also check the influence of the number of drones, clients and warehouses on the profit the company can make.

1.2 Hypotheses

We formulate the following questions and provide our hypotheses for the answers to them.

1. *Is there a difference between CNET, CNCP and DynCNET in the profit the company can make?*
   We expect that the company can make more profit using DynCNET than using CNCP, which should be more profitable than CNET.

2. *Is it possible to make a profit using any of the three approaches?*
   We expect that all three approaches should be profitable given the right number of warehouses, drones and clients.

3. *What is the influence of the number of drones on the profit?*
   We expect the profit to scale linearly with the number of drones. Once all clients start to get delivered, saturation will set in.

4. *What is the influence of the number of warehouses on the profit?*
   We expect the profit to scale with the number of warehouses. As more warehouses become available, it will become more likely to have a warehouse with a low price for the product near the client.

5. *What is the influence of the number of clients on the profit?*
   We expect the profit to rise until there are too many clients, after which the fines of not delivering start to become too large.

6. *Is there a difference in average delivery time between CNET, CNCP and DynCNET?*
   We expect that CNCP and CNET will deliver faster than DynCNET (as DynCNET wastes time when it switches). We don't expect a difference between CNCP and CNET as both will start delivering quickly and won't waste any time by switching.

7. *Is there a difference in the number of clients that are not delivered between CNET, CNCP and DynCNET?*
   We expect that CNET and CNCP will deliver the same amount of clients and DynCNET less, again because of the switching between clients.

8. *Is there a difference in the number of messages that needs to be exchanged between CNET, CNCP and DynCNET?*
   We expect that CNCP will need less messages because we expect less rounds of bidding. CNET will need more messages but still less than DynCNET, which needs a lot of messages due to the dynamism to make sure the best contract is being executed.

This report is organized as follows. Section 2 discusses the theory of the three protocols that are tested. Section 3 gives the multi-agent system design of our problem and compares it to the theory. Section 4 details the results of the experiments that were run to test the hypotheses. Section 5 concludes with a discussion of the hypotheses.

2 Theory

In this section we describe the theory of the Contract Net Protocol and the two extensions CNCP and DynCNET. We use the literature on these protocols to highlight their characteristics and differences. We also discuss some alternative extensions to CNET.

2.1 Contract Net

The Contract Net Protocol, or CNET, was conceived by Smith in 1980 [8]. It is a high-level protocol for communication between agents to allow distributed task execution. The high-level protocol messages have a particular structure and meaning, which means the agents can interpret them to receive information on tasks and allocations and to send replies such as bids.

CNET tries to solve the problem of assigning tasks to idle agents in the most efficient way. Moreover, this problem needs to be solved in a distributed manner, without a central authority that tracks tasks and assigns them with global knowledge.

CNET relies on the negotiation of contracts for the execution of a task between a manager and a contractor. The manager (or initiator) launches a tender for the execution of a task and will monitor the execution and process the results once they arrive. The contractor (or participant) bids on the tenders and when it wins, executes the task and submits the results to the manager. Each agent can be one of these roles for each individual task. This can also mean that an agent acts as a contractor for one task and as a manager for sub-tasks of this task to distribute the workload over several agents.

The Contract Net Protocol is defined by the messages and the information that is contained within them. By having a common language for these messages, all agents can communicate with each other and new agents can immediately start listening for new messages and participate in the negotiation process. There are three types of messages: those related to task announcements, those related to bids and those for the reporting of task results.

A manager initiates the contract negotiation process by advertising a task with a task announcement message. This message can either be broadcasted to all agents or sent only to one or a few agents in a focused manner. The task announcement message contains four main parts: an eligibility specification, a task abstraction, a bid specification and an expiration time. The eligibility specification describes what criteria an agent must meet to be able to bid and eventually possibly receive the contract and execute the task. The task abstraction is a description of the task to be executed. The bid specification outlines the expected form of a bid. The expiration time marks the final time at which a bid may be submitted.

Whenever a task announcement arrives at an agent that wants to act as a contractor, all of these parts of the announcement allow contractors to decide whether and when to submit a bid. They add the announcement to a list of received and not expired announcements if they're eligible. The announcements in the list are ranked on interest to the agent. This ranking is a task-specific operation and is based on the information contained within the task announcement.

An agent cannot bid while busy, but a busy agent continues to process the task announcements while executing its task. When this task completes, it can then check the list of task announcements and pick a task to submit a bid on. An idle agent will decide to bid on a task either immediately when it arrives or when it expires. It either submits a bid to the most interesting task or waits for new (better) task announcements. In a bid, an agent describes its specification in the node abstraction slot. It can also indicate additional information that will be required if the agent may execute the task. Whenever an agent submits a bid to a task, it allocates some resources to that task. This is called early commitment. That agent can then not bid on other tasks at the same moment. This is done to avoid too many task awards to one agent, which would all need to be queued and would slow down the system performance by putting too much load on one agent, which means tasks get executed later.

Bids arrive at the manager and are also ranked and stored in a list until the task can be awarded. The manager can either award a contract immediately if a satisfactory bid arrives (even before the expiration time) or wait for further bids. When the task announcement expires, it can either award the contract to the best bidder in its list or advertise the task again (immediately or after waiting some time). The task is therefore not necessarily awarded when it expires. If a manager awards a contract, it sends a message to the winning contractor, who can start executing the task. When the contractor is finished, it sends a report to the manager with the results of the execution.

The bidding process can be extended to allow agents that cannot or do not want to bid to respond immediately with their reason why they will not bid. Managers can then delay task announcements if no agent is ready to take up the task. An agent can respond immediately if they are ineligible, busy or give the task a low ranking. The manager can then avoid simply reissuing the task announcement when it expires, but can loosen the eligibility requirements if all agents are ineligible or wait before reissuing the task announcement if all agents are busy or not interested yet in the task.

The FIPA version of the Contract Net Protocol [2] differs from the original protocol by adding acceptance and rejection of a bid by a manager. This means a contractor is notified if its bid was rejected.

2.2 Contract Net with Confirmation

The Contract Net with Confirmation Protocol, or CNCP, was conceived as an extension to the Contract Net Protocol by Knabe et al. in 2002 [6].

The major shortcoming of CNET that CNCP tries to address is the early commitment by agents. If multiple managers announce a task at the same time, a contractor is only allowed to bid on one of these tasks, after which it already allocates some resources to that task. The agent will choose the task that seems most interesting at that time. The problem is that if a task is somehow very popular, many agents would bid on it but only one would be awarded the task. The other agents then have to wait for the next opportunity to restart the bidding process and try to receive a task. Furthermore the agent has allocated some resources to that popular task, which possibly are only freed after the task announcement expires. This means the agent has less available resources to bid on other tasks later on.

To resolve this issue, in the CNCP the full commitment to a task is moved to the awarding phase: if a contractor is awarded a task, only then does it allocate its resources and stop bidding on other tasks. While a bid has been submitted but the task has not been awarded yet, the contractor can continue bidding on other tasks. When a manager wants to award a task to a contractor, it first contacts that contractor with the request to accept the task. This contractor may however already been awarded a different task, in which case it will refuse the task. The manager may then try to award the task the next best contractor. This either continues until a contractor accepts the task, in which case the contractor can start executing the task and the manager can send a rejection message to the other bidders, or if no contractor accepts it, the manager can announce the task again (possibly after waiting some time).

CNCP requires two more messages to allocate a task: one to request the execution of a task by the manager and one reply to this message by the contractor. If the first contacted bidder refuses, the next best bidder has to be tried and two more

messages need to be sent. In the worst case this means $2n$ extra messages (with n the number of bidders) must be sent.

2.3 Dynamic Contract Net

The Dynamic Contract Net Protocol, or DynCNET, was conceived as an extension to the Contract Net Protocol by Weyns et al. [9, 10, 11].

DynCNET tries to improve on CNET by adding dynamism to the allocation and execution of tasks. More specifically, it addresses the fact that an agent usually has to do some preparation when it is assigned a task before it can actually start executing this task. DynCNET allows contracts to be switched during this preparation phase.

DynCNET changes the acceptance of task by a winning contractor to a provisional acceptance. This means that until further notice that contractor will execute the task and report the results. The contractor can then start its preparation for the execution of the task. During this preparation phase however, the contractor is free to continue bidding on other (new) tasks and the manager may continue announcing the task. If either receives an offer that they consider better than the current contract, they are allowed to cancel the current contract and pursue another task. As soon as a contractor finishes its preparation and starts the actual execution of the task, the contract is final. The contractor is fully committed to the task and no longer bids on other tasks. The contractor also notifies the manager that it is executing the task, which means the manager stops announcing the task.

The synchronization between managers and contractor to assess their current state requires some additional messaging. If a manager cancels a contract, it needs to notify the original contractor that the task has been awarded to another agent. Inversely, a contractor that takes up another task needs to notify the manager of the original task that it is no longer committed to executing that original task. Finally the contractor needs to send a message to the manager when it starts executing the task, so the manager can stop announcing the task to other agents.

2.4 Alternative extensions

In literature, other extensions of CNET can be found. [1] describes a protocol that divides the bidding phase in two: pre-bidding and definitive bidding. The first round, pre-bidding, is only tentative. When bids have been received, instead of awarding the task to a participant the initiator informs all the participants of the tentative results. Based on these tentative results and on how the world and other running bidding processes evolve, participants can still change their bids. Once a participant has been tentatively accepted, it can send a definitive bid to the initiator.

If this bid is still better than all the tentative bids from the other participants, the participant will be awarded the task and both agents commit to the contract. All other participants will be informed that the bidding process is finished. If the definitive bid is not the best bid, the participant gets tentatively or definitively declined. In case of the former, it can retry to get the task by submitting a new tentative bid.

This protocol can be seen as an extension on CNCP. Participants who get accepted, can use the definitive bid to accept or decline: they will submit a high bid in the former case and a low bid in the latter. The advantage of this protocol over CNCP is that participants who initially get rejected can still improve their bid.

[3] describes a similar protocol: iterated contract net (ICN). When an initiator receives bids, it can choose to accept a bid, or can iterate the process by issuing an adapted task announcement to (a subset of) the bidders. The intention of this next iteration is to receive better bids. The protocol stops when a participant is awarded the contract, the initiator stops issuing calls for proposals (because all bids are unsatisfactory) or all participants stop bidding.

One of the main differences between [1] and ICN is that in ICN new task announcements are issued, while the same task announcement is reused in [1]. An initiator in the ICN protocol can thus actively try to increase the price of the contract.

[7] uses game theory to prove that a leveled commitment is to be preferred over full commitment in all situations. In the negotiation of a contract in this protocol, a decommitting penalty is agreed, which might be different for both agents. When an

agent wants to cancel the contract, it then has to pay this decommitting fee to the other agent. The fee becomes a factor in calculating whether or not it is interesting to cancel the current contract in favor of another contract.[1] It also allows for contracts that were not possible using full commitment. Normally the benefit from an offered contract could be lower than the expected benefit from other contracts that have not yet been announced, which means the agent would wait and not commit to that offered contract. By having the right decommitting fees, this offered leveled commitment contract could become interesting and could be taken up.

As these protocols are not part of our objective, we have not implemented our design for them.

3 Multi-agent system design

3.1 Design

We consider a logistics problem where a company serves an area with a fleet of drones. In this area, a number of clients appear with an order for a specific product. One of the drones takes up the order and is dispatched to a warehouse to collect a package with that product. Finally it flies to the client to deliver the package. The goal is to maximize the profit earned on making these deliveries.

Across the world several warehouses are located. For simplification, these warehouses have an infinite stock of every package that a client could potentially order. Drones can pick up packages at these warehouses for a price that varies per warehouse in a range of 80 to 120% of that package's market price. These warehouses are divided over the whole area in such a way that a clustering of warehouses is avoided as much as possible. To achieve this, the number of warehouse is split into powers of two (e.g. $5 = 4\ (2^2) + 1\ (2^0)$). For every power of two x, the whole area is divided in x equiareal rectangles, each containing one warehouse at a uniformly random position.

Drones can unexpectedly crash and fail. The drone is then written off and the cost of a drone is subtracted from the profit. Drones are also constrained by their batteries, they only have a limited range before they need to charge. Whenever the battery level gets very low, the chances of crashes increase, as the power that comes from the batteries becomes less and less stable. Drones can charge their batteries at the warehouses for a certain cost. They will intelligently choose a route when delivering a package to try as much as possible to avoid having a low battery.

A client has only one order and when that order is delivered, the client stops participating. A client with multiple orders can be reduced to this model by creating multiple clients with one order each. The order consists of the type of package a client wants, the end time by which it wants to have received the package, the price it is willing to pay and the fine it levies if the package is not delivered before the end time. The price and the fine vary around the market price. There are both clients that are initially present in the world and clients that appear randomly afterwards, uniformly distributed over time.

For simplification, all agents are completely aware of the whole world and can send messages to all other agents with 100% reliability.

The failures of drones are based on a Poisson distribution and are used in calculating the bid that drones will make on tasks. The bid is based on the estimated minimal cost for a drone to serve the client. The estimated cost consists of four parts: the cost to buy the package from a warehouse; the cost of charging for the battery drain caused by moving between the drone's current location, the warehouse, the client and back to a warehouse to charge; the probabilistic cost of a drone failure on that trajectory; and the price the client is willing to pay (which is subtracted from the other costs). The drone will decide for which warehouse this combined cost is the lowest and submit that as its bid, on the condition that the cost is lower than the fine (then it would just be better not to bother delivering), that it can arrive with some battery power left and that it can deliver before the end time of the order.

[1] The paper proves, using the chicken game from game theory, that the consideration by an agent of canceling a contract is more than just checking if the added profit of the new contract is more than the decommitting fee that would then have to be paid. The other agent might decommit at some point, which means that this agent gets released from the contract without paying a fee and even receives the fee from the other agent.

3.1.1 Agent state machines

Both the drone and client use finite state machines to define in which phase of the contract negotiation they are situated.

Figure 1 shows the FSM for the drone agent in the three different protocols[2]. The drone starts in the `IDLE` state. In this state, it will move to the closest warehouse, where it will enter the `CHARGING` state and start charging its battery. In CNET and CNCP, a drone can bid in the green states. In DynCNET, a drone can bid in the green and yellow states. In CNET, a drone will enter the `PROPOSED_BID` state once it has submitted a bid. It will reenter the `IDLE` state if the client declines. In CNCP and DynCNET, the drone stays in either `IDLE` or `CHARGING` and may continue bidding.

If the client awards the task to the drone, the drone goes into the `PICKING_UP` state. There it will start moving to the warehouse for which the estimated cost was the lowest. When it gets there, it will go to the `CHARGING_FOR_CONTRACT` state and start charging until either it is sufficiently charged or until it is time to leave due to the client's delivery deadline. In DynCNET, the drone may continue bidding in the `PICKING_UP` and `CHARGING_FOR_CONTRACT` states. If it wins another contract, it will stay in or go to the `PICKING_UP` state. When the drone leaves, it will pick up the package from the warehouse and enter the `DELIVERING` state. From then on, it will keep flying to the client until it has arrived, upon which it will deliver the package to the client and enter the `IDLE` state, thereby moving to the closest warehouse.

The `PROPOSED_BID` state is only used in CNET, not in CNCP nor DynCNET. DynCNET also has two extra transitions: from `PICKING_UP` and `CHARGING_FOR_CONTRACT` back to `IDLE`, in case the client cancels the contract. CNCP and DynCNET also have a direct `gotContract` transition from `IDLE` and `CHARGING_FOR_CONTRACT` to `PICKING_UP`. In CNET the drone first has to pass through the `PROPOSED_BID` state.

Figure 2 shows the FSM for the client agent in the three different protocols. The client starts in the `AWARDING` state. When its task has been awarded, it enters the `ASSIGNED` state. From then on, the client will go to the `EXECUTING` and `DELIVERED` state respectively when the drone picks up and delivers the package. Only a drone failure or the cancellation of a contract in DynCNET can change the state flow, by having the client return to the `AWARDING` state. If the maximum delivery time for the client has passed before a drone delivered the package, the client will enter the (final) `OVERTIME` state.

In CNET and CNCP, a client can only send task announcements while in the `AWARDING` state. In DynCNET, it can also send task announcements in the `ASSIGNED` state, until the package is picked up by the drone and the client enters the `EXECUTING` state.

3.2 Comparison with existing MAS from literature

When clients enter the world, they will act as managers and broadcast their task announcement to all agents.[3] This task announcement consists only of the order that a client has. All drones are considered to be eligible. As there is only one type of bid, the bid specification defined in CNET is implicit and left out of the task announcement. There is no separate expiration time for bidding: drones are expected to bid immediately and the task announcement will be sent at the latest on the end time of the order.

In CNET, only one bid per opportunity (tick) is allowed, so the drone bids on the task for which its estimated cost is the lowest of all received task announcements. Afterwards the drone is committed to that task and cannot bid on other tasks until it is either awarded the task and execution is complete, or until the client explicitly refuses the bid. In the original CNET protocol, a drone would be allowed to bid again at the next tick. Although [8] suggests that not allowing this might cause significant delays in the system, due to the fact that tasks do not have a bidding expiration time, that clients immediately award contracts if possible and that drones only have one resource, i.e. they can only serve one client at a time, we feel that it does not

[2]Not represented in the figure is the failed state, in case the drone crashes. A drone can enter this state from all other states and can never leave it again.

[3]This includes other clients, who will ignore all task announcements. RinSim does not (yet) provide a broadcast to a subset of all agents, in this case the drones.

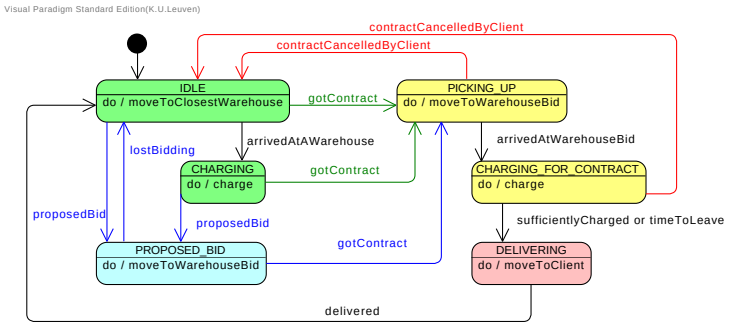

Figure 1: A finite state machine of the drone agent. The blue transitions and state only apply to CNET, the red transitions only to DynCNET and the green transitions to CNCP and DynCNET.

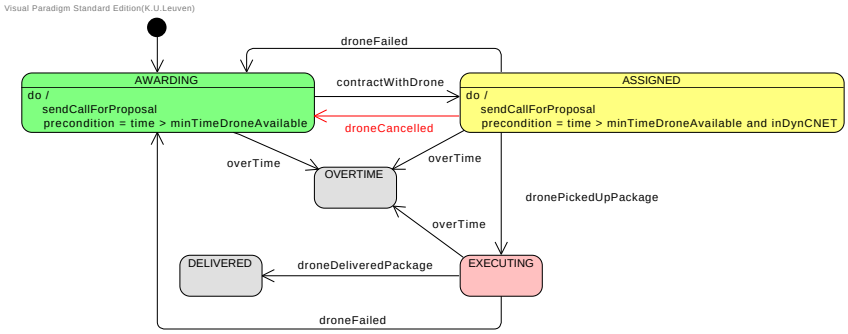

Figure 2: A finite state machine of the client agent (the red transition only applies to DynCNET)

have as significant an effect. In CNCP and DynCNET, the drone bids on every incoming task while it is not committed to a contract.

The extension where drones that do not bid can respond with their reason is also supported. If a drone has no warehouse for which they can deliver the package for a cost below the fine, they declare themselves as ineligible. If a drone is delivering a package, it answers that it is busy. If in CNET there is a valid bid for a task but it is not the lowest cost of all task announcements, it replies that it has given the task a low ranking.

[8] suggests that drones could declare when they are available and what kind of tasks they can execute, so clients can send proposals to award a task directly to an eligible drone. We have decided to instead add a timestamp to the message that the drone is busy, which is a lower limit of when the drone might become available again, based solely on the current distance to the client and not taking the distance to and from warehouses into account. The client can use this information to delay task announcements if no drone is available or interested. More specifically, if all drones are busy or ineligible, the minimum of all estimated times when drones could become available again is taken as the next time for the task announcement. This reduces the number of messages that are sent by the client (and replies by the drones). If a drone gives the task a low ranking, the task announcement is immediately retried as other more interesting tasks will have been awarded and the remaining drones may now rank this task as the best.

A client waits one tick for bids to arrive. All bids are satisfactory, as drones only bid on tasks that are economically viable. The task is then awarded to the best bidder, i.e. the drone with lowest cost. It informs the winning drone that they may start executing the task, and notifies the other bidding drones that their bid has been refused. In CNET, this means the drone can start bidding on other tasks again. Bids that would somehow arrive later will also be refused by the client.

In CNCP and DynCNET, drones can submit

multiple bids at the same time and could potentially be awarded multiple tasks. As a drone can only execute one task at a time, it chooses the task with the lowest cost, notifies the client and starts executing it. It denies all other task assignments and notifies those clients.

The other awarded tasks are not queued as in the original CNET protocol, but released for other drones to fulfill, as delivery takes too long for queuing to be sensible. Clients whose assignments have been denied will reannounce the task instead of trying the next best bidder as in the original CNCP, as chances are high that due to the short time between announcements and bidding, this next best bidder has already been assigned another task.

In the original CNCP the acceptance of a task by a drone is a two-step process: the client first requests that a drone accepts a task, the drone then replies whether they can accept or not and only then does the client formally accept the drone's proposal, award it the task and commit to the contract. This process is merged in our design into one message that the client is willing to accept the proposal. The drone can then refuse if necessary. While the original CNCP therefore requires $O(2*n)$ more messages, our design only requires one extra message per bidding round. The number of bidding rounds is however increased in comparison with the original CNCP protocol.

Once a drone has received a task, it starts executing the task by flying to the warehouse with the lowest estimated cost (that was calculated when bidding). It picks up the package at the warehouse and can possibly charge its battery there, either until it must leave or until the battery is sufficiently charged to minimize the risk of failure. It then flies to the client, delivers the package and flies to the closest warehouse to charge and wait for new tasks. The client is sent a message when the package is picked up and when it is delivered.

In case the drone would crash, the notification of this failure to the client is modeled by sending one last signal to the client. In practice, outside a simulator, a polling approach might be used. After the client is aware of the failure, the client announces the task again.

A potential risk of DynCNET is oscillation of tasks between drones. [9] describes a solution by limiting the areas of interest, e.g. by broadcasting a task announcement only within a limited range. In our design, the agents are limited only by their battery, but not with a communications range. We did not take any special action to prevent the oscillation of tasks between drones.

4 Experiments

To test our hypotheses in our multi-agent system design, we experiment using implementations of this design for the three protocols.

4.1 Setup

We ran two experiments. In the first one, we varied the number of drones, warehouses, initial clients and *extra clients*, i.e. the clients that appear randomly after the start of the simulation.

After running the first experiment, we noticed that DynCNET has a breakdown when there are many clients compared to the number of drones, as figures 6 and 7 show. The second experiment therefore aims at exploring these circumstances by running an experiment with a large number of drones and a bigger range for the number of clients.

The values for the independent variables for both experiments can be found in table 1 (for succinctness, the protocol is excluded from the table). Multiple dependent variables were measured in these experiments. These can be found in table 2. The values of the other variables, that were not varied and were chosen to be as realistic as possible, can be found in table 3.

All experiments were run using RinSim 4.1.0 on computers at the P&O lab at the Department of Computer Science. These computers have a 3.4 GHz quad-core processor and 8 GB of RAM. For each combination of values for the independent variables, 150 simulations were performed[4] to rule out that one random experiment could skew the results. All raw data has been kept and can be found in attachment to this report.

[4]For a total of 1417500 and 351000 simulations for respectively experiment 1 and 2.

Variable	Experiment 1	Experiment 2
# drones	1..10	10
# warehouses	4..10	10
# initial clients	5..25 step 5	5..100 step 5
# extra clients	10..50 step 5	10..200 step 5

Table 1: Independent variables and their ranges (with inclusive boundaries)

Variable	Meaning
totalProfit	The profit the company made (in €)
nbClientsNotDelivered	The number of clients that weren't delivered
averageDeliveryTime	The average delivery time in milliseconds (for the clients that are delivered)
nbMessages	The number of messages exchanged
averageNbSwitches-PerDrone	The (average) number of times a drone switched contracts

Table 2: Dependant variables and their meaning

Variable	Value
Prices packages	€100, €200, €500, €1000
Speed drone	50 km/h
Range battery	15 km
Size world	10km x 10km
Failure lambda drone	0.1 (< 10% battery), 0.01 (< 20%), 0.0001 /km
Price drone	€3000
Cost energy	€2/km
Pay-off client	between 1.2 and $1.3 * marketPrice$
Fine (undelivered) client	$0.5 * marketPrice$
Delivery window client	between 1000 and 5000 seconds (after first announcement)
Last client enter time	1 hour
Battery charging rate	50s to fully charge

Table 3: Other variables and their values

4.2 Results

Figures 3 to 11 show the results of our experiments. The error bars represent the 95% confidence interval.

4.3 Analysis

To test whether the experimental data we obtained proves or disproves our hypotheses, we used the Games-Howell test. As the data is heteroscedastic, the TukeyHSD method will have too many Type I errors [5]. [4] showed that Games-Howell is also robust with non-normal data, which is the case for some of our results. A significance level of 95% was used for all tests.

Figure 3 makes it clear that the number of drones has an influence on which protocol maximizes profit. When there are many drones (7 or more), DynCNET is better than CNCP, which outperforms CNET. For an average number of drones (4 to 6), CNCP outperforms CNET and DynCNET. For 3 drones or less, there is no difference between CNET and CNCP, but both outperform DynCNET. As figure 6 and 7 show, no breakdown happens when enough drones are present. DynCNET is thus only able to outperform CNCP and CNET with enough drones. Otherwise, it performs worse. CNCP outperforms CNET for an average or large number of drones.

As figures 3–6 show, a company can indeed make a profit using any of the three protocols, given the right number of drones and warehouses for the given number of clients in the world.

When increasing the number of drones, the profit also increases until enough drones are present in the world and saturation is achieved. As saturation sets in quickly in our experiment, no region where the profit increases linearly could be found.

When increasing the number of warehouses, the profit increases as well. Figure 4 shows that this increase seems to slow down as the number of warehouses is further increased. More experiments would have to be conducted to fully test this hypothesis.

When increasing the number of clients (figure 11), the profit first increases as well. When the number of clients increases even further, the

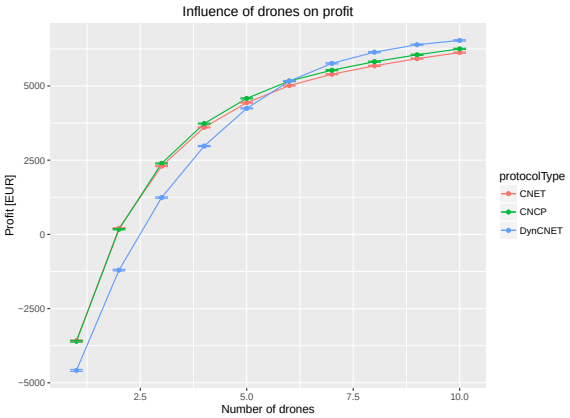

Figure 3: The influence of the number of drones on the profit

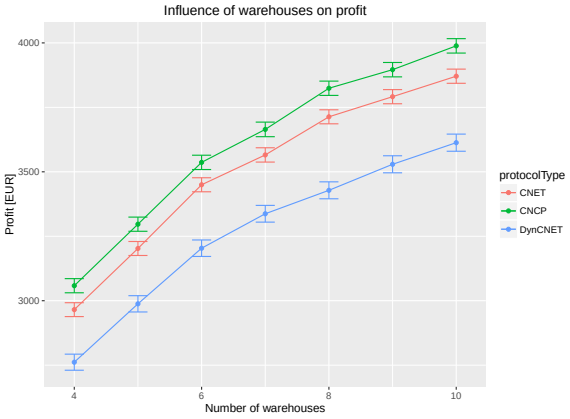

Figure 4: The influence of the number of warehouses on the profit

Figure 5: The influence of the number of clients at the start of the simulation on the profit

Figure 6: The influence of the number of clients that appear after the start of the simulation on the profit

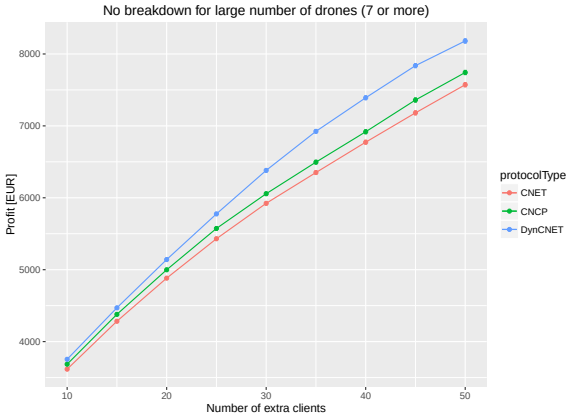

Figure 7: The influence of the number of clients that appear after the start of the simulation on the profit, with a large number of drones

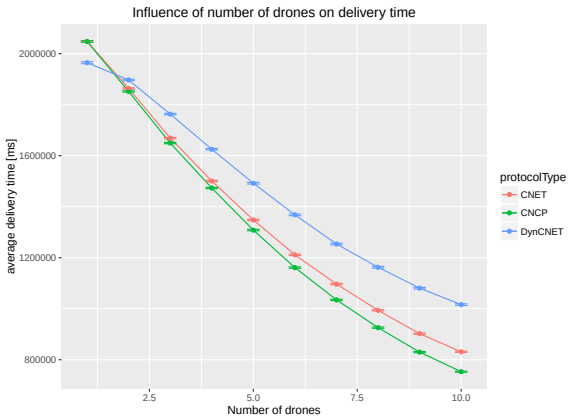

Figure 8: The influence of the number of drones on the delivery time

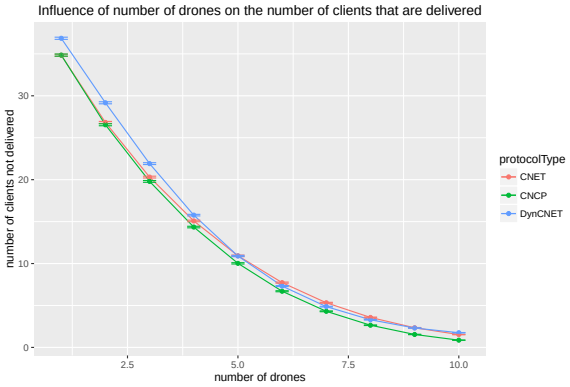

Figure 9: The influence of the number of drones on the number of clients that are not delivered

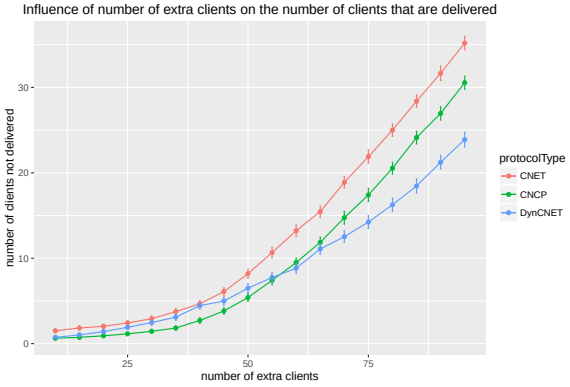

Figure 10: The influence of the number of clients that appear after the start of the simulation on the number of clients that are not delivered (in the second experiment, for 25 initial clients)

Figure 11: The influence of the number of clients that appear after the start of the simulation on the profit (in the second experiment)

number of clients whose packages aren't delivered also increases (figure 10). This means the profit starts to fall, as the extra fines start to dominate the extra profits. DynCNET outperforms CNET and CNCP in the number of clients that get delivered when the number of clients increases. However the fall of the profit for DynCNET is a lot steeper, which means in DynCNET actually less profit is earned than in CNET and CNCP. This can be explained by the number of times a drone switches their contract and thereby wasting time turning around. This number increases linearly with the number of clients up to an average of 15 switches per drone when 200 clients appear after the start of the simulation. Taking actions to prevent oscillations might pay off in the case of a large number of clients.

Figure 8 shows the influence of the number of drones on the delivery time. For 1 drone, DynCNET delivers the fastest, while there is no significant difference between CNET and CNCP as both protocols don't behave differently if there is only one drone. If two or more drones are present in the world, CNCP will deliver faster than CNET, which will deliver faster than DynCNET.

The impact of the protocol on the number of clients that don't get their package delivered also depends on how many clients and drones there are (figures 9 and 10). For a small number of clients (less than 20 clients initially and less than 50 clients after the start) or a large number of drones (10 drones), CNCP will deliver the most packages, followed by CNET and DynCNET. In other scenarios in the first experiment, CNCP will still outperform the other two protocols. For 9 drones, there is no significant difference between CNET and DynCNET. But when there are 8 drones, DynCNET outperforms CNET. When the number of drones decreases further (to 4 drones or less), CNET outperforms DynCNET again. In our second experiment, DynCNET outperforms CNCP when the number of clients increases (figure 10). However, in our experiments CNET never outperformed CNCP. At best, it can keep up with CNCP when there is only 1 drone due to similar behavior.

When there are a small number of clients (5 initial clients and 10 extra clients), no significant difference in the number of exchanged messages could be found between CNET and CNCP. However, in

other configurations with more clients, CNCP actually needs more messages than CNET (not less). DynCNET always needs more messages than the other two protocols.

5 Conclusion

Our objective in this report was to investigate CNCP and DynCNET, two extensions of CNET, and compare them to CNET in a drone delivery setting where drones deliver packages from warehouses to clients. To be able to do experiments for this comparison, we researched the three protocols and implemented them for our logistics problem, adapting them to fit the specific setting of drones, warehouses and clients.

We ran two experiments on these implementations: one where the number of drones, warehouses and clients were varied, and one where the number of drones and warehouses were fixed and the number of clients was varied. Using these experiments, we can now evaluate our hypotheses.

The profit that is earned is influenced by the protocol used, but depends on the specific configuration of drones, warehouses and clients. We expected DynCNET to be more profitable than CNCP, and CNCP more profitable than CNET, which is the case but only when there are many drones. For lower number of drones, CNCP improves on CNET or performs equally well, but both are better than DynCNET. This is due to a breakdown in DynCNET when there are too many clients for too little drones. All three approaches are able to generate a profit if the number of drones and warehouses are appropriate for the number of clients.

When the number of drones increases, the profit also increases until a point of saturation. This saturation sets in quickly so it cannot be said that the increase in profit is linear. When the number of warehouses increases, the profit also increases. The increase slows down when the number of warehouses is further increased. When the number of clients increases, the profit increases at first before it starts to fall again as the extra fines start to dominate the extra profits.

Except when there is only one drone, CNCP delivers faster than CNET which delivers faster than DynCNET. DynCNET is slower due to task switching, but CNET is also slower than CNCP, contrary to what we expected.

While we expected CNCP and CNET to deliver the same amount of clients, in practice, CNCP is clearly better than CNET. In some situations, DynCNET is actually better than CNET and for a large number of customers, even better than CNCP.

As expected, DynCNET needs more messages than the other two protocols due to messaging continuing when both agents are not fully committed to the contract they have. CNCP however needs more messages than CNET when there are a large number of clients. For a small number of clients, CNET and CNCP need about the same number of messages.

In conclusion we could say that overall CNCP and DynCNET are an improvement on CNET. Between the two extensions the choice for our setting is less clear cut. The largest factor that determines the relative performance is the proportion of clients to drones. DynCNET performs better when there are enough drones so that all clients can be delivered and there is little competition between clients for the drones, i.e. the number of times a drone switches contracts is low. When too many switches start to occur, CNCP becomes the better protocol.

References

[1] Samir Aknine, Suzanne Pinson, and Melvin F. Shakun. An extended multi-agent negotiation protocol. *Autonomous Agents and Multi-Agent Systems*, 8(1):5–45, 2004.

[2] FIPA TC Communication. *FIPA Contract Net Interaction Protocol Specification.* Foundation for Intelligent Physical Agents, 2001. Doc. SC00029.

[3] FIPA TC Communication. *FIPA Iterated Contract Net Interaction Protocol Specification.* Foundation for Intelligent Physical Agents, 2001. Doc. SC00030.

[4] Paul A Games, Harvey J Keselman, and Jennifer J Clinch. Tests for homogeneity of variance in factorial designs. *Psychological Bulletin*, 86(5):978, 1979.

[5] Mia Hubert. *Statistische modellen en data-analyse*. Acco, 2015.

[6] Tore Knabe, Michael Schillo, and Klaus Fischer. Improvements to the FIPA contract net protocol for performance increase and cascading applications. In *In International Workshop for Multi-Agent Interoperability at the German Conference on AI (KI-2002)*, 2002.

[7] Tuomas Sandholm and Victor Lesser. Leveled-commitment contracting: A backtracking instrument for multiagent systems. *AI Mag.*, 23(3):89–100, September 2002.

[8] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, Dec 1980.

[9] Danny Weyns, Nelis Boucké, Tom Holvoet, and Bart Demarsin. DynCNET: A protocol for dynamic task assignment in multiagent systems. In *First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*, pages 281–284, July 2007.

[10] Danny Weyns, Nelis Boucké, Tom Holvoet, and Bart Demarsin. DynCNET: A protocol for flexible transport assignment in AGV transportation systems. Report CW 478, KU Leuven, Department of Computer Science, 2007.

[11] Danny Weyns, Nelis Boucké, Tom Holvoet, and Kurt Schelfthout. DynCNET: A protocol for flexible task assignment applied in an AGV transportation system. In Barbara Dunin-Keplicz, Andrea Omicini, and Julian A. Padget, editors, *Proceedings of the 4th European Workshop on Multi-Agent Systems EUMAS'06, Lisbon, Portugal, December 14-15, 2006*, volume 223 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.