Sytronix - Energioptimering med asynkrona motorer och variabla pumpar

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH Ingenjörshögskolan vid Campus Helsingborg
Industrial Electrical Engineering & Automation, IEA, LTH

Författare:
David Barbulovic
Adam Zeidan

Examinator: Mats Lilja
Handledare: Bjarne Restedt

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Institutionen för Industriell elektroteknik och automation
Lunds universitet
Lund 2014

Sammanfattning

I dagens moderna industri där hydraulik används, sätts alla motorer i drift på sin nominella arbetslast, där pumparna är fasta. Flödet som ges regleras med hjälp av ventiler som stryper flödet. Från ett energimässigt perspektiv bidrar denna metod med stora energiförluster, då efterfrågan kan vara mycket lägre än tillgångarna. Vi ser i dagens samhälle hur energikostnaderna stiger frekvent, vilket har gjort att man idag ger mycket mer uppmärksamhet och tar större hänsyn till denna faktor.
Motorerna styrs med drivsystem som bygger på master/slave-principen, där mastern beordrar slavarna. Problemet med dagens system är att om en masterstyrenhet försvinner, så kommer systemet att kollapsa, vilket har gjort att man idag vill utveckla ett system där alla drivenheter kan fungera som potentiella masters eller slaves.

I detta examensarbete fokuserar vi på analys och undersökningar av möjliga lösningar till både ett stabilt oberoende system och en smart energioptimering, där vi har som mål att fördela arbetsbelastningen mellan motorerna på ett så energieffektivt sätt som möjligt. Examensbetet görs hos Bosch Rexroth i deras kontorslokaler.

I rapporten presenteras vår analys av lösningsmetoder, ge teoretiska och praktiska exempel för att motivera våra beslut, samt kommentera vilka nackdelar, fördelar dessa lösningar bidrar med. Det är viktigt att notera att alla lösningsförslag i analysen inte kommer att tas med som resultat. Detta kommer behandlas i diskussionskapitlet.

Drivenheterna som används i detta examensarbete är Bosch egna och har namnet Indradrive. Styrenheten innehåller både en PLC och en PID-regulator. Tanken är att implementationen ska ske genom Bosch egna mjukvara Indraworks. Implementationen sker endast i mån av tid och kommer endast vara en prototyp för två asynkronmotorer.

Nyckelord: examensarbete, effektoptimering, asynkronmotor, drivsystem, variabla pumpar, automation.

Abstract

In modern industry where hydraulic is used, all the motors run on their nominal workload, where the pumps are fixed installed. The given flow is regulated with the help of valves which chokes the flow. From an energy perspective this contributes to a great amount of energy losses, where the demand can be much lower than the supply. We can observe from today's society that the energy costs rise frequently, which have contributed to more attention and greater consideration to this factor.
The motors are controlled by a drive system which is built on the master/slave principle, this means that the master commands the slave. The problem with today's system is that if a master disappears, then the whole system will collapse. This has created an urge to develop a system where all the drivers can be a potential master or slave.

In this thesis our main focus is on the analyzing and research parts of possible solutions to both a stable, independent system and a smart energy optimization, where our main goal is to achieve an energy efficient work distribution between the motors as possible. The thesis work is done at Bosch Rexroth in Helsingborg.

We will present our analys of the solution methods in this report and give both theoretical and practical examples, to motivate our decisions, comment advantages and disadvantages to what these solutions contributes with. Notice that all our solutions in the research part will not be included in the result, we will discuss this later in the discussion part.

The drivers which are thought for use in this thesis are Bosch own and are named indradrive. It contains a PLC and a PID-regulator. If time permits the implementation will be done in Bosch's software Indraworks as a prototype for asynchronous motors.

Keywords: Thesis, effect optimization, asynchronous engines, drive systems, variable pumps, automation.

Förord

Vi vill först och främst tacka Bosch Rexroth och speciellt Bjarne Restedt, för att ha gett oss möjligheten att utföra vårt examensarbete hos dem. Detta har bidragit till vår personliga utveckling och utvidgat våra kunskaper.

Ett stort tack till Andreas Lago för hjälpen och rådgivningen vi fått ta emot under examensarbetet.

Vi vill även rikta ett stort tack till Mats Lilja för en god handledning. Vi kommer sakna dem roliga stunderna och trevliga samtalen vi haft under åren.

Stort tack till våra familjer som trott på oss och stöttat oss under denna långa tid.

Innehållsförteckning

1 Inledning

1.1 Företagshistoria

Företaget Bosch grundades av Robert Bosch år 1886, en tysk ingenjör, som var bland annat känd för att utveckla tändstift till bilmotorer. Han var bland annat en av de första som introducerade åtta timmars arbetsdag. Företaget har vuxit sig stort och omfattar flera områden i industrin. I dagsläget kommer cirka 80 % av inkomsten till företaget från bilindustrin. [2]

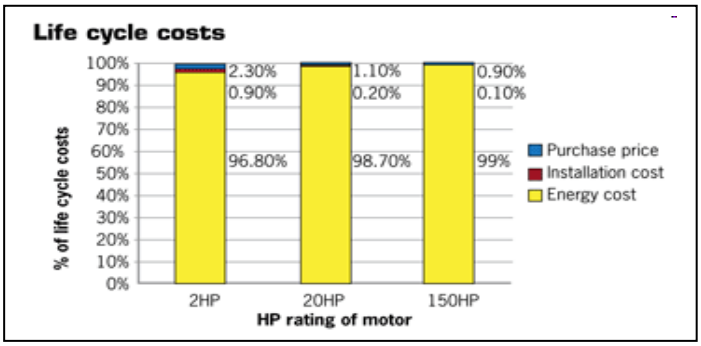
1.2 Allmänt om dagens industri

I processindustrin finns en stor uppsättning drivsystem för pumpar och fläktar. Dessa system kräver i allmänhet ingen sofistikerad styrning. Däremot finns anledning att i dagens läge införa bättre varvtalsreglering för de många pumparna och fläktsystemen för att spara energi. Den mängd energi som kan sparas genom varvtalsreglering i stället för strypning i dessa tillämpningar motsvarar för Sveriges del storleksordningen något kärnkraftverk. [1]

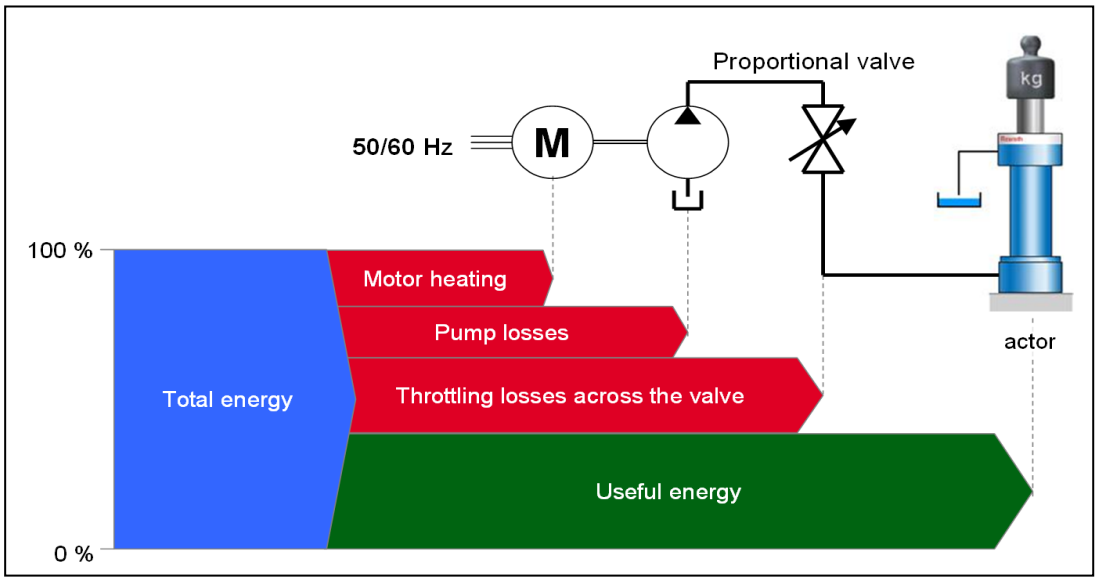
Många pumpar och fläktar drivs idag utan varvtalsreglering. En enkel och gammal metod som används för att minska luftflödet är strypning med ventiler. Trots att metoden är mycket energislösande är den fortfarande mycket vanlig. [1]

Antag t.ex. att det verkliga luftbehovet är 80 % av fullast. Luftflödet är ungefär proportionellt mot fläktens hastighet. Om varvtalet kan regleras till 80% av maxvärdet motsvarar detta en effektreduktion till $(0,8)^3 \approx 0,51$ med andra ord till *hälften*. Tar man dessutom hänsyn till att friktionsförlusterna minskar, så blir energibesparingen stor. [1]

Man brukar räkna med att av livstidskostnaderna för en större pump svarar inköp och installation för 5 %. Underhållskostnader utgör ytterligare 5 %. Resten, 90 %, är energikostnader. Potentialen för energibesparing genom varvtalsreglering av elektriska maskiner är enorm. Trots detta är medvetenheten om detta besvärande låg. [1]

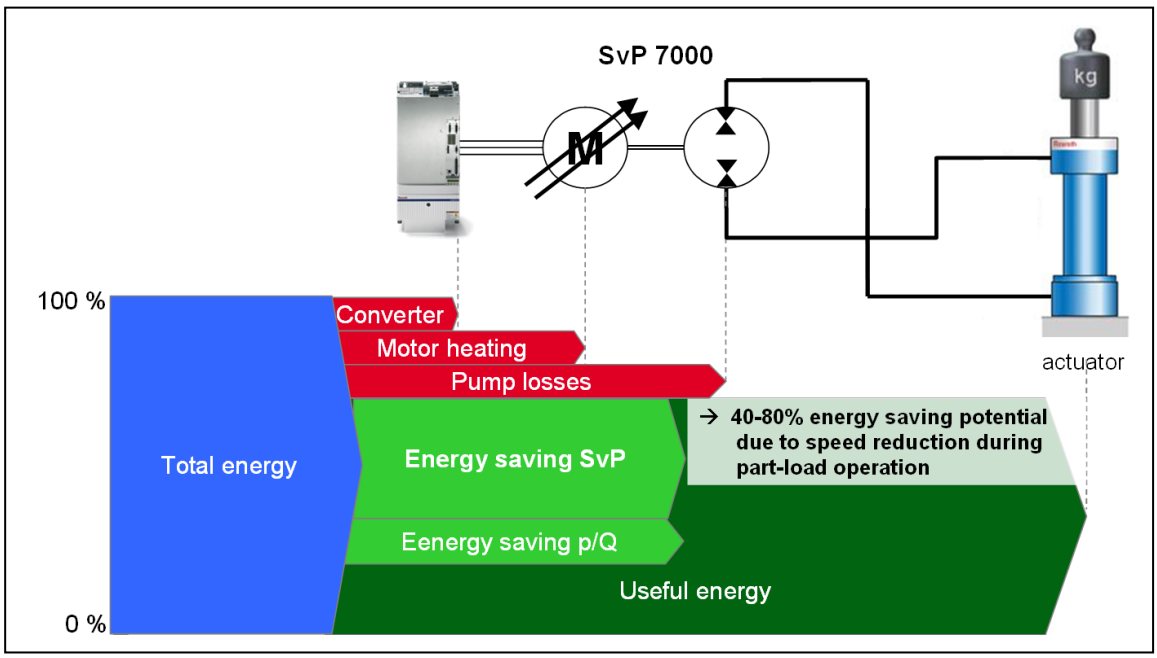


Figur 1.00 [6]. Diagram över energikonsumtion.



Figur 1.01 [6]. Traditionell reglering med dåligt energiutnyttjande.

Med Sytronix är tanken att man ska spara energi genom varvtalsreglering, med en driver, innehållande en PLC, PID-regulator kopplad med en asynkronmotor med en variabel pump.



Figur 1.02 [6]. Framtida reglering med förbättrat energiutnyttjande.

1.3 Problemformulering och Bakgrund

Projet, som är en tillverkare av vattenskärare, har gett Bosch Rexroth i uppdrag att utveckla ett system där alla drivrar, anslutna till variabla pumpar, kan kommunicera med varandra med ”Master-Slave” principen. Men den här principen ska **inte** baseras på Profibus, utan alla drivrar i systemet ska ha

potentiallen att bli en **temporär** master eller slav, beroende på vilka kriterier som möts.

Dagens industri använder sig nämligen av kommunikationsstandarden Profibus, men har en stor nackdel. Skulle Mastern för systemet försvinna, kollapsar systemet.

Om en ny slav tillkommer, måste en ny lista skapas hos mastern och andra konfigurationer måste göras för att göra mastern medveten om den nya slaven. Detta har gjort att man efterfrågar ett system där drivenheter är oberoende av varandra och självständiga. [3]

En annan fråga är hur ett system med flera variabla pumpar, ska kunna arbetsfördela arbetsbördan emellan sig, på ett energieffektivt sätt. Systemet ska ha egenskapen att endast starta de motorer som behövs för att möta tryckbörvärdet. I och med att det är tänkt att använda vattenskärare, är det oerhört viktig att behålla det önskade trycket. Uppnås inte detta, får man ett flöde som inte penetrerar igenom materialet. Det är viktigt att notera att det önskade trycket ändrar sig kontinuerligt inom loppet av sekunder, eller i minuter.

1.4 Mål

Följande mål sattes uppp innan arbetet påbörjades:

1. Ett av delmålen är att undvika överdimensionering hos motorn/motorerna. Dvs, att de arbetar med en last som inte är alltför liten jämnfört med sitt nominella värde.
2. Starta och stänga av motorer så energieffektivt som möjligt.
3. Varje drivenhet ska ha potentialen att vara temporära master eller slav
4. Minimera antalet motorer i drift.
5. Ta hänsyn till arbetsmiljön, genom att reglera motorn för att ha en så låg ljudnivå som möligt.
6. Implementera systemet i mån av tid.

2 Metod

Eftersom det här systemet inte existerar idag, hade vi möte varje vecka med vår handledare. Där presenterades idéer för problemlösning, som handleraren och hans kollega gav respons på. Genom detta, lärde vi oss en del ytterligare faktorer som spelar större roll. Samtidigt lästes kursliteratur och användes kunskaper från kursen Elmaskiner och Drivsystem.
För att få inblick i begränsningar hos systemet studerades manualer om Indraworks. För att underlätta tänkandet har vi använt oss av teoretiska beräkningar och illustrationer.

3 Teknisk Bakgrund

3.1 Termiska Förluster

För att konstruera elektriska drivsystem räcker det inte med lämpliga momentkurvor eller adekvat effekt. Termiska transienter från oundvikliga värme- och friktionsförluster i motor och axlar är mycket betydelsefulla i drivsystem. Olika maskiner har olika isoleringsmaterial och tål därför olika temperaturer. En maskin klassas således efter den värmetålighet den har. [1]

Exempelvis betyder isolationsklass att den övre gränstemperaturen tillåts vara 130^{o}C medan klass F tillåter 155^{o}C. Detta kommer att avgöra vilket driftsätt som är möjligt för den aktuella maskinen, t ex kontinuerlig drift, snabba starter och stopp e.d. [1]

De vanligaste värmekällorna i ett drivsystem är

- kopparförluster
- järnförluster
- friktionsförluster

Kopparförluster uppstår i lindningar, kablar, borstar och kommutatorer. Detta härrör från resistansen i ledningarna och är proportionella Ri^2. Eftersom det elektrodynamiska momentet är proportionellt mot strömstyrkan i rotorn kan man kvantitativt påstå, att stora moment åstadkommer stora kopparförluster.
Kopparförluster kan vara besvärliga att leda bort med kylflänsar o.d. eftersom lindningarna ligger långt inne i maskinen och dessutom är inkapslade i isolermaterial. [1]

Järnförlusterna uppstår i magnetiskt material som utsätts för varierande magnetfält. Flödesvariationerna inducerar spänningar som driver s.k. vivelströmmar. Dessutom flyttas de manetiska domänerna vilket ger friktionsförluster i materialet som kallas för hysteresförluster. Som visats tidigare är hysteresförlusterna proportionella mot ytan av B-H-kurvan. Man strävar därför efter av skapa material med så smal hystereskurva som möjligt. För att minska virvelströmsförlusterna delar man i materialet i elektriskt isolerade delar vilka hindrar virvelströmmarna att ta sig fram. Det vanligaste sättet är att laminera med plåtar av magnetiskt material vilka är belagda med isolerande lackskikt. [1]

Friktionsförlusterna kommer från lager, borstar, kylfläktar och från friktionen mot den luft eller gas som omger rotorn. Det senare blir ett allvarligt problem vid extremt snabbgående maskiner eller maskiner med stor radie varför man ibland fyller dessa med en ”lågfriktionsgas”, t.ex. helium. [1]

De olika värmeförlusterna beror på ett komplext sätt av maskinens varvtal och belastning, liksom av strömmarnas och spänningarnas amplitud och form. [1]

3.2 Asynkronmotorn

Asynkronmotorn är den i särklass mest använda motorn i industrin. Asynkronmotorn används i alla möjliga sammanhang, till exempel för att driva fläktar, pumpar och transportband. De allra flesta asynkronmotorerna används i drifter med konstant varvtal. Ny kraft- och mikroelektronik har ökat möjligheten att varvtalsreglera asynkronmotorn vilket ytterligare har vidgat motorns användningsområde, exempelvis till traktion. Asynkronmotorer finns i alla effektklasser, från watt till megawatt. [1]

Att asynkronmotorn används i så stor utsträckning beror framför allt på följande:

- Den är självstartande när den ansluts till nätet
- Den är robust och pålitlig
- Den är billig i inköp
- Den är enkel och billiga att underhålla
- Den är starkt standardiserad

Sista punkten är viktig. Standardiseringen avser bland annat motorns dimensioner, fästordningar (montering på fot eller fläns), kylning (vätske- eller luftkylning), startmoment och effekt. Det för att motorer av olika fabrikat är direkt utbytbara. Motorn blir på så vis billig och enkel att få tag på.[1]

Asynkronmaskinen kan liksom de flesta elektriska maskiner även fungera som generator. Den används huvudsakligen i mindre kraftverk (vatten- och vindkraftverk) med effekter upp till ett par megawatt. [1]

Generellt gäller att motorn får högre verkningsgrad vid sin nominella belastning ju större den är. I en stor motor finns det mer plats för koppar vilket gör att resistansen och resistansförlusterna relativt sett blir mindre. Luftgapet är också relativt sett mindre i en stor motor. Detta innebär dock inte att man alltid sparar energi genom att välja en större motor. Förlusterna i en överdimensionerad motor som inte belastas fullt kan mycket väl bli större än i en mindre motor med nominell belastning. [1]

3.3 Drivsystem och Profibus

Man använder drivsystem, drivenheter, för att kunna styra en/flera enheter. Dessa tillåter programmering i PLC. För att få flera enheter att prata med varandra, använder man sig av Profibus, som är standardiserad i industrisammanhang.

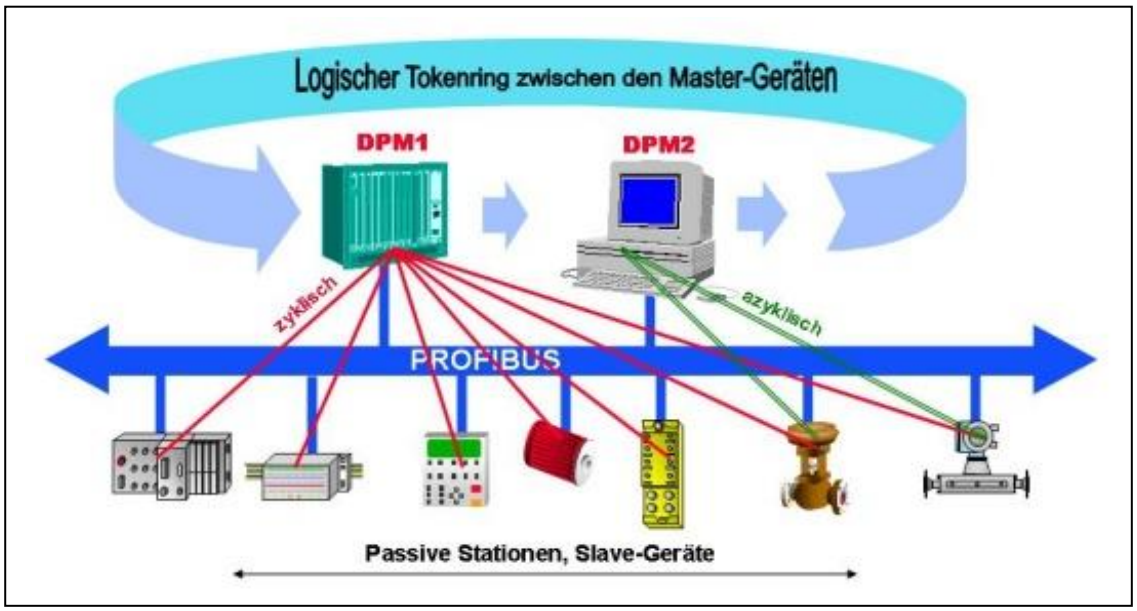
Det finns två typer av Profibus, men vi nämner endast denna, det standardiserade “PROFIBUS DP” kommunikationsprotokollet. Det stöder en stor variation av applikationer i fabriks- och processautomation såväl som motorstyrning och felsäkra applikationer. [4]

Det är designad för seriella I/O-enheter, i automationsindustrin.[3]

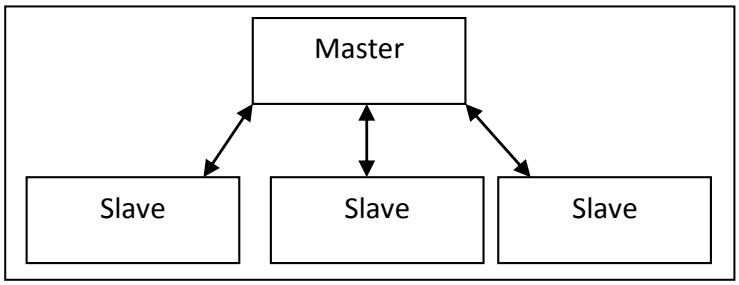
Systemet, Profibus, använder sig av en master och slav-enheter. En Profibus-slav kan vara allt från I/O transduktor, ventiler, nätverks drivenhet och andra mätapparater. Slaven tar emot data och skickar utdata till mastern, den kan endast ”acka” mottagna meddelanden, eller skicka respons meddelanden till mastern vid förfrågning. Det är viktigt att notera att all nätverkskommunikation initieras av Mastern.[3]

Det finns två olika klasser av masters, klass ett master hanterar vanlig kommunikation eller utbyte av data med de slavar som är tilldelad till den. En klass två master är en speciell enhet som används för drifttagning av slavar och diagnostisering. [3]

En klass-1-master är normalt sett en PLC eller en PC som exekverar en speciell mjukvara. Medan klass två master enhet är oftast en konfigurations enhet, kan vara en laptop eller programmeringsbar konsol. Man använder den mest till att ge användaren informationen om systemet, men används även för drifttagning, underhåll eller diagnostik. Kommunikationen mellan klasserna initieras av en master av klass två. Notera att endast en master ge utsignaler till slavarna och broadcasta alla anslutna slavar eller individuellt. [3]



Figur 3.01 Profibuss [5]

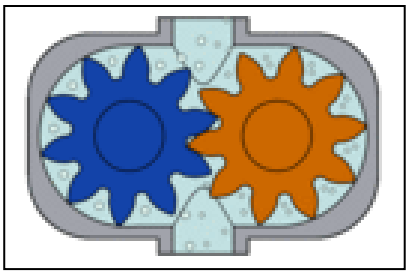


Figur 3.02. Förenklad bild av Profibuss.

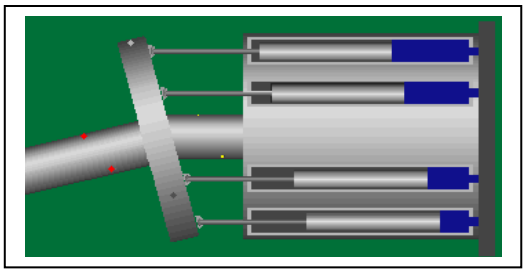
3.4 Konstanta- och Variabla pumpar

Vilken skillnad är det konstruktionsmässigt mellan fasta och variabla pumpar? Förträngningspumpar som är fasta, utgörs ofta av en hydraulisk pump där konstant deplacement volym V1 per varv och därmed ett konstant flöde Q1 vid konstant körhastighet n1: Q1 = V1 * n1, dvs. de främjar alltid samma flödeshastighet. Flödeshastigheten beror på hastigheten av pumpen. [6]

I den variabla pumpen är volym per cykel variabel. Fördelen med en variabel är dess låga effektförluster, eftersom det ger bara så mycket flöde som behövs i varje enskilt fall. Skillnaden ligger således i den variabla flödeshastigheten för den variabla pumpen i motsats tillden fasta pumpen. [6]



Figur 3.04 Fasta Pumpar[6]

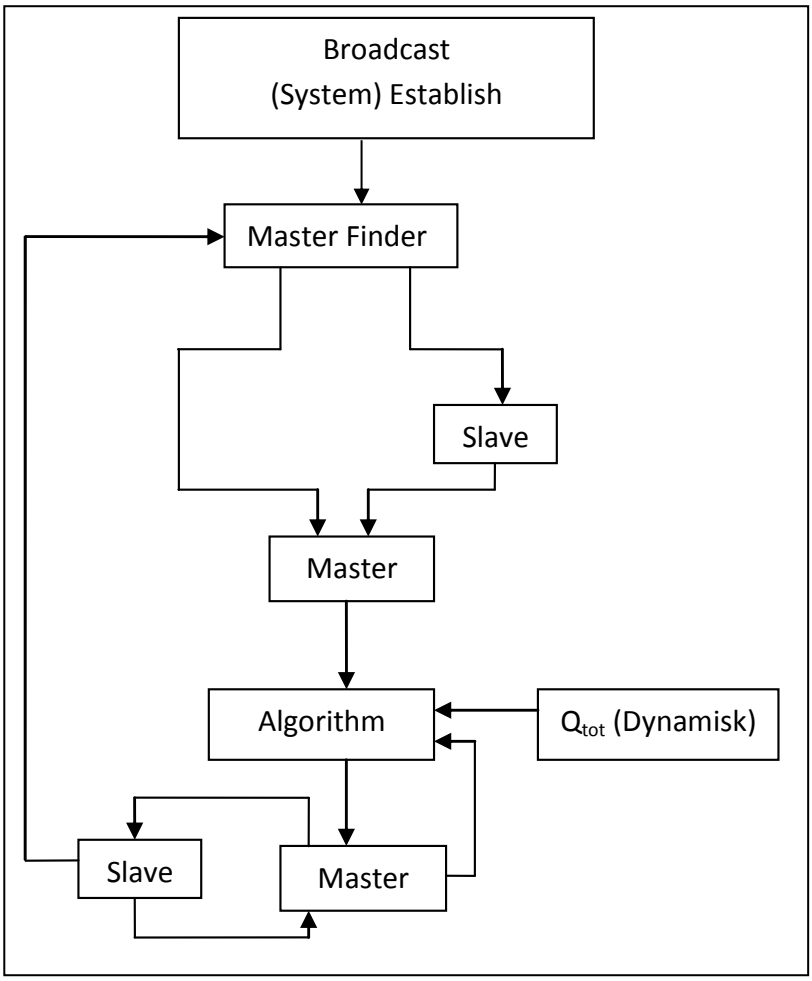


Figur 3.05 Variabel Pump [6]

4 Analys

4.1 Struktur för kommunikationen av systemet

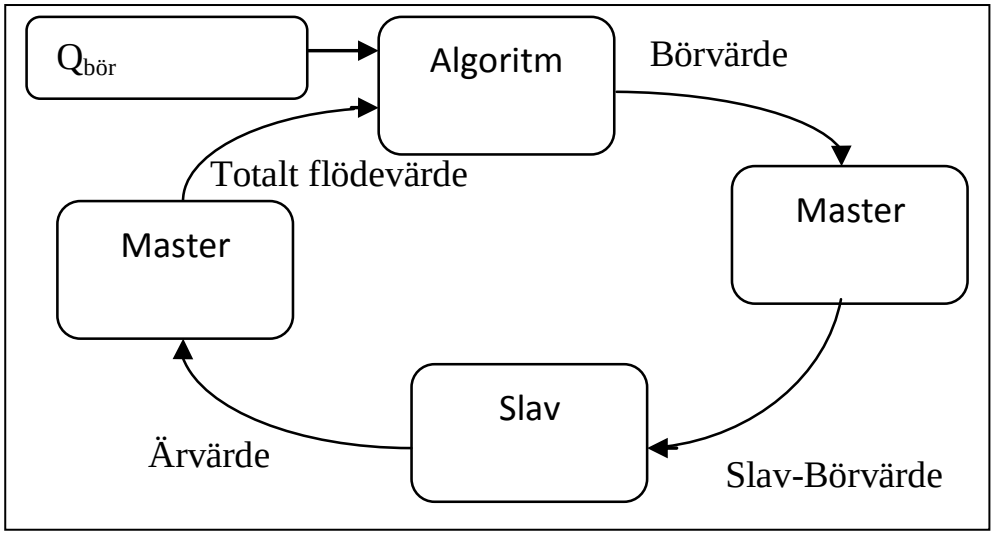
Arkitekturen för systemet är framtagen genom analys för hur systemet måste fungera rent logiskt. Först visas den övergripande strukturen för systemet och därefter förklaras stegvis delarna i systemet.



Figur 4.01. Teoretisk skiss för systemet.

Det allra första som sker är att alla drivenheter utför sin Broadcasting och Master/Slaves utses. Slavarna skickar sin data till Mastern som får information om varje enhets maximala och minimala flödeskapacitet, $Q_{\alpha,\ max}$ och $Q_{\alpha,\ min}$, därefter använder mastern sig av algoritmen med ett dynamisk flödesvärde för systemet, $Q_{bör}$, för att kalkylera fram ett börvärde som passar individuella drivenheter. Mastern använder dessa värden till att skicka till Slaven som i sin tur skickar tillbaka sitt ärvärde. Med ärvärde menar vi ärvärdet för varvtalet (n), flödet (Q) och trycket (p). Om Mastern försvinner, utses en ny Master av Slavarna och om en Slav försvinner, utför Mastern algoritmen och beräknar utan Slaven som har försvunnit. Viktigt att notera är att det totala efterfrågade flödet $Q_{bör}$, ändrar sig kontinuerligt. Systemet måste även kontinuerligt

beräkna ett nytt börvärde för varje drivenhet, man kan säga att systemet loopar. Bilden nedan visar detta (Figur 4.02).



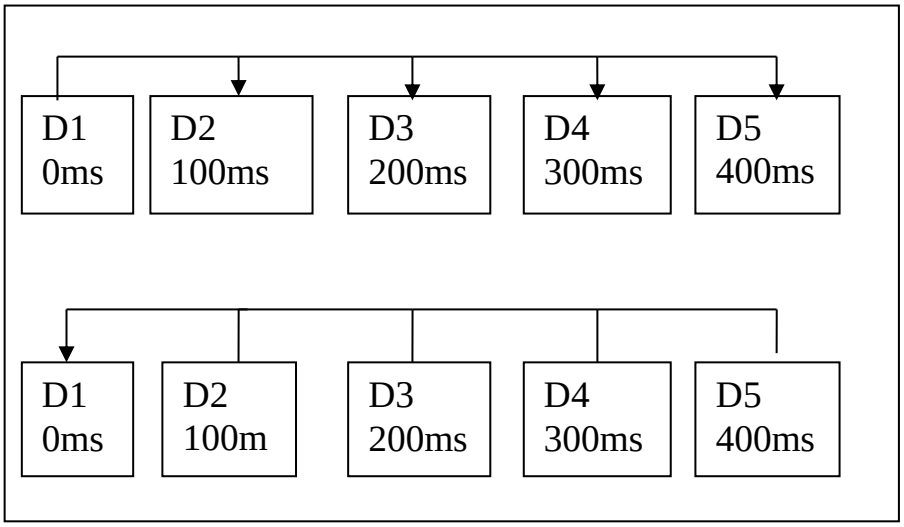
Figur 4.02. Signalfödet i systemet.

4.2 Etablering av kommunikation för Systemet och Master/Slave

Ett av delmålen i examensarbetet är att drivenheter i vårt system är att ska alla kunna vara potentiella Masters och Slaves. För att göra detta måste vi etablera kommunikation emellan alla drivenheter. Men hur gör vi detta? Enligt vår handledare är kriterierna att vi använder oss av UDP-protokollet i ett subnät, samt hade man tänkt sig att man etablera kommunikationen mellan drivenheter med hjälp av tidfördröjning, som är förprogrammerat för varje drivenhet. För varje drivenhet ökar tidsfördröjningen med 100 millisekunder. Efter denna fördröjning startas drivenhetens broadcast och IP-adresserna sparas i varje drivenhets respektive register.
Vi beslöt oss dock att undersöka om man skulle kunna automatisera detta istället föra att ha en tidsfördröjning på varje drivenhet förprogrammerat. Tanken var att man skulle använda sig av en id som varje drivenhet automatiskt skulle tilldela sig, men problemet blir då hur en enhet ska kunna veta vilken id de andra har? Problem kan då uppstå t.ex. genom att två enheter har samma nummer. Detta ledde till att denna idé förkastades.

UDP-protokollet är tänkt att användas istället för TCP-protokollet. Orsaken till detta är att om vi använder oss av TCP, måste vi etablera en server med klienter, vilket orsakar ytterligare tidsfördröjningar, då man måste etablera en server och skapa en kommunikationslänk mellan varje klient. Dessutom är alla drivenheter i ett subnät, vilket är onödigt då drivenheterna i intranätet (subnätet) inte behöver kommunicera med omvärlden (internet). UDP passar bättre eftersom paket då skickas direkt till klienterna istället för att varje enhet ska behöva agera som server. Problemet med UDP är just att man inte kontrollerar att mottagaren har mottagit paket, men detta problem är löst, då

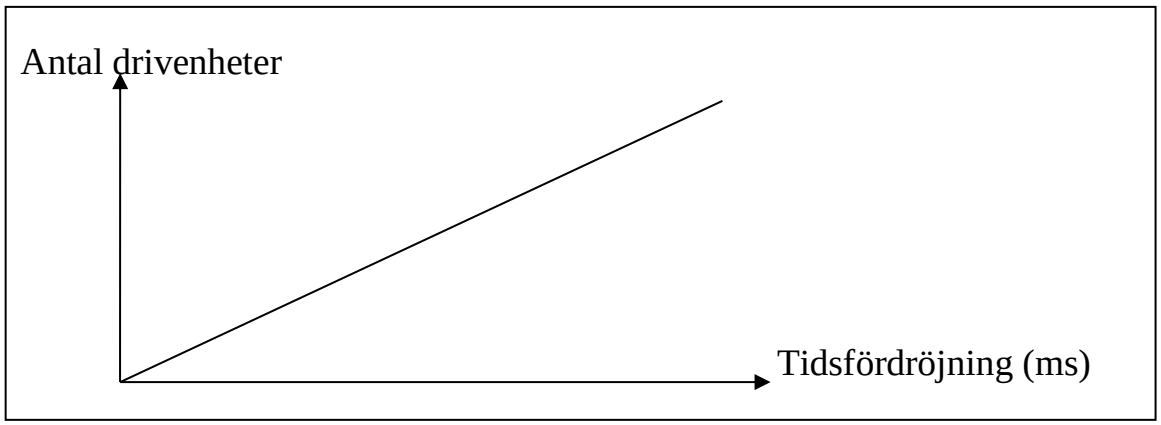
programmet Indraworks gör det möjligt att man skapar ett UDP-system där mottagaren måste skicka ett acknowledgement för varje mottagen paket. Tanken med tidsfördröjningen är att man skapar en sekventiell etablering av kommunikationen i systemet. Varje gång en drivenhet skickar sin broadcast, ska den registrera de mottagande enheternas IP-adresser. Följande illustrerar hur detta fungerar för en viss tidsfördröjning.



Figur 4.03. Initiering av anslutning mellan drivenheter.

Drivenhet ett skickar sin broadcast, därefter skickar drivenheterna D2, D3,D4 och D5 en respons till D1, som registrerar deras IP adresser i D1:s register. Efter 100ms börjar D2 skicka sin broadcast och får svar från D1, D3, D4 och D5, som registreras i D2:s register. Detta sker fram till den sista drivenheten, som är i vårt fall D5.

Problemet som tidsfördröjningen medför är att tiden för att etablera kommunikationen i systemet ökar linjärt med antalet drivenheter i systemet. Detta medför att man får en begränsning. Hur länge är användaren villig att vänta för att kommunikationen ska etableras? Om användaren har sex drivenheter, medför detta att man får en tidsfördröjning på 500 ms, dvs. en halv sekund. Den första drivenheten behöver inte ha någon tidsfördröjning. Men om användaren istället väljer ha 50 enheter, medbringar detta stora tidsfördröjningar då 50 drivenheter ger oss en tidsfördröjning på fem sekunder. Vi ska inte glömma att vi också ska välja Master, utföra kalkyleringar och få Mastern med Slavarna i arbete. Användaren skulle kunna specificera maximal väntetid han vill ha och är redo att vänta.



Figur 4.04. Teoretiskt diagram över anslutningstiden mellan drivenheter.

När kommunikationen är etablerad, ska Mastern och Slavar utses **beroende** på deras IP adresser. Drivenheten med den lägsta IP adressen blir den tillfälliga Mastern, resten blir Slavar och man går vidare till nästa steg i systemet.
Ett viktigt problem att notera här är att antalet enheter, som är tänkt att etablera kommunikation med, är okänd. Systemet kan etablera ett kommunikationsnät som inte alla drivenheter är med i, då vissa kan vara felkonfigurerade eller icke kontaktbara. En lösning hade kunnat vara att varje enhet förproggrammerades med hur många enheter den ska kunna kommunicera med.

Om vi förprogrammerar dessa begränsar vi systemet, maskinoperatören kommer inte att kunna installera in en ny drivenhet i systemet, då alla enheter är redan förprogrammerade för ett begränsat antal. Denna problemlösning bidrar till ytterligare problem, vilket gör att vi förkastar den.
Den enklaste problemlösningen, hade varit om vi helt enkelt visar i displayen, vilka som är anslutna i systemet, IP adress, samt vem som är Master/Slav. Ansvaret läggs på operatören som via informationen på displayen kan kontrollera vilka enheter som är anslutna, vilket även underlättar felsökning

4.3 Master Gone

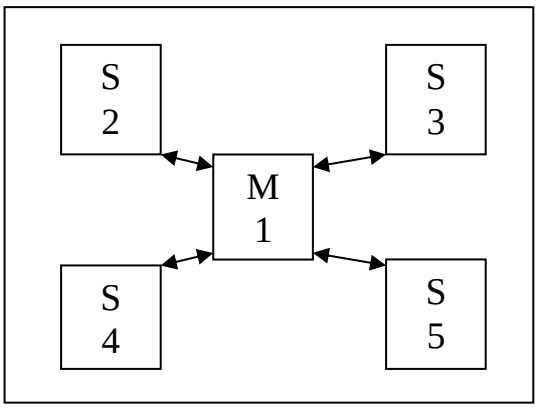
Grundtanken med Sytronix, är att alla drivenheter ska vara oberoende av varandra samt potentiella Masters. Själva Mastern bär det yttersta ansvaret för att kalkylera ut arbetsbördan för varje drivenhet, kontrollera att alla slavar är aktiva samt om en ny drivenhet tillkommer.
Om Mastern skulle försvinna, får systemet inte stanna, då vi missar målet med Sytronix. Mastern, i vårt fall, har det yttersta ansvaret som nämns ovan, men en del av ansvaret för att systemet ska fungera ligger också hos Slaven.

Slaven måste kontrollera om Mastern fortfarande själv är aktiv. Man kan göra detta genom att låta Slaven ha ett ”Time-For-New-Master” (TFNM), notera att

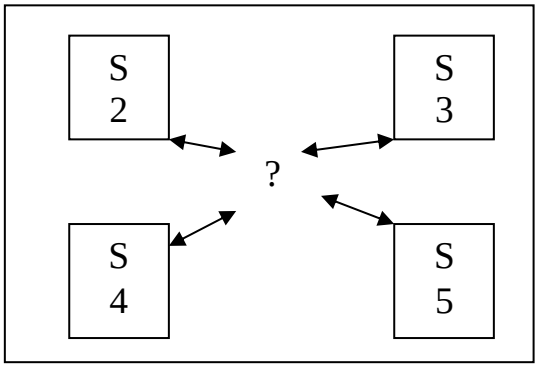
detta är ett namn för en funktion som bör skapas. Det som menas med TFNM är att man låter slaven innehålla någon sorts av nedräkning, kan jämföras med Time-to-Live för TCP-protokollet, för hur länge en Master inte skickar någon data/ping till slaven. Varje gång slaven mottar data/ping från Mastern startas TFNM om på nytt. Har inte Slaven mottagit något från Mastern under en viss tidsperiod, aktiveras TFNM och en ny Master utses. Problemet med detta är att alla drivenheters klockor måste vara synkade, vilket de är.

Hur ska den nya Master utses? När själva systemet startade, valdes den första Mastern genom att ta den lägsta IP adressen i systemet, varje drivenhet i subnätet skickade ut en broadcast, efter en viss försening, och registrerade det i sitt register. Den utsedde Mastern skickar kontinuerlig broadcast för att kontrollera att alla drivenheter finns kvar i subnätet. Slavarna har kvar detta register och kan via den utse en ny Master.
Varje Slav kontrollerar vilken IP adress som är lägst, därefter utses en ny Master, som då mottar data från sina slavar, som då initierar Algoritmen.

Ex:

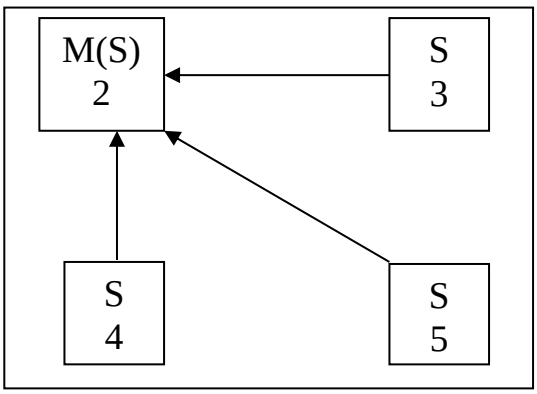


Figur 4.05. Mastern är kvar.



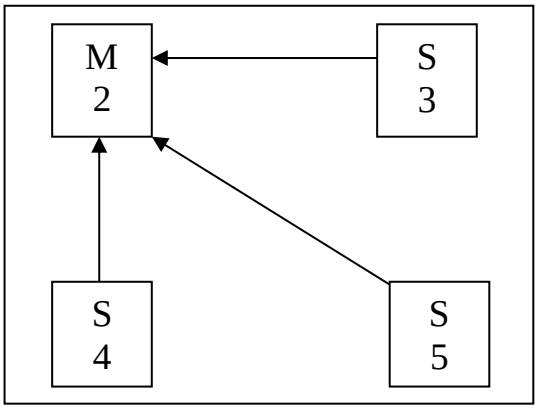
Figur 4.06. Mastern försvinner.

TFNM sätts i alla slavar till fem sekunder i vårt exempel. Efter att TFNM har passerat den utvalda tidsperioden, aktiveras funktionen. När den aktiveras, kommer alla slavar att individuellt söka efter en ny Master, baserat på vilken IP adress de har i sitt register. I vårt fall, har Slaven med IP adressen två, den lägsta adressen i subnätet vilket gör den till Master.
Viktigt att notera, vilket värde TFNM bör ha, är okänd i dagsläget, då detta inte har testats i verkligheten. Tiden för TFNM bestäms genom att undersöka medeltiden för hur lång tid det tar för Mastern att kalkylera fram och skicka $Q_{bör}$ order till sina slavar.



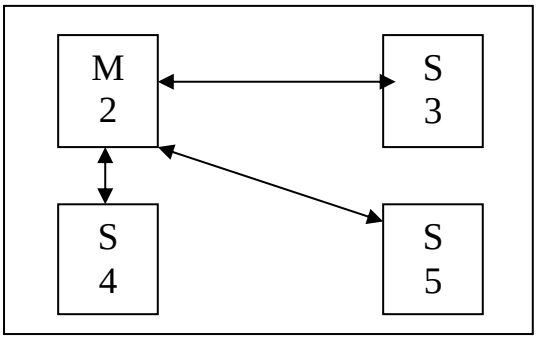
Figur 4.07. Slaven S2 blir Master.

Mastern skickar ut en broadcast för att kontrollera att alla Slavar i systemet är kontaktbara. Slavarna måste i sin tur skicka data till Mastern ($Q_{max,}$ $Q_{min,}$ eventuellt fler data). Som den registrerar för varje IP adress.



Figur 4.08. Drivenhet nummer två (Master), mottar data från sina slavar.

När Mastern har fått in all data från alla Slavar initierar den Algoritmen och skickar order till individuellt till Slavarna. Efter det är vi på samma punkt som innan den första Mastern försvann.



Figur 4.09. Master skickar ut ordrar som slavar i sin tur skickar ärvärden.

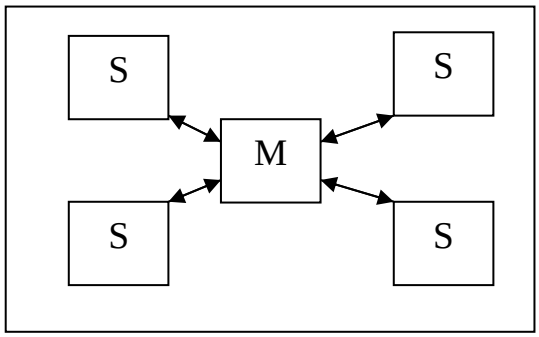
Något som bör tas upp är om en Slaves register bör uppdateras, kapitlet New Driver tar upp detta. Notis: Fallen Slave Escaped och Master Gone, bör ha den cykliska kontrolltiden initierad efter att en Master har blivit utsett.

4.4 Slave Escaped

Det kan hända att kunden måste serva en motor och väljer därför att stänga av den, eller att en drivenhet oturligt nog inte längre fungerar. Som resultat måste slavarna, inklusive Mastern, få en ökad arbetsbörda för att mötta kundens behov. Ansvaret för detta ligger hos Mastern, då det är den som utför arbetsfördelningen. Den måste kunna detektera felet, frågan är då hur ska den göra det?
Som vi diskuterade innan i broadcasting delen, måste det finnas en cyklisk tid hos Mastern som pingar alla slavar i nätet, om slavarna finns där kommer de att svara till Mastern och på så sätt kan Mastern kontrollera om de är aktiva i systemet.
Om Mastern inte får ett acknowledgment från slaven/slavarna, ska detta ses som att slaven/slavarna inte längre är aktiva och raderas från sin ”registret”, därefter initieras Algoritm funktionen igen och räknar endast med de i

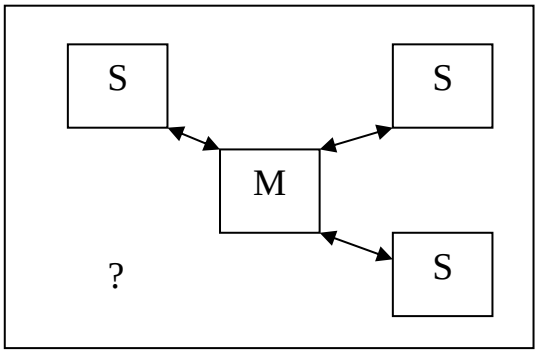
registret. Mastern bör då informera operatören via displayn att slavarna med IP adress/adresser x inte längre är aktiva. Detta ger kunden möjligheten att se vilken drivenhet som inte svarar och eventuellt felsöka.

Ex. Alla maskiner är i drift, Mastern kontrollerar med en cyklisk tid om drivenheterna är aktiva.
Mastern pingar sina Slavar och mottar acknowledgements:



Figur 4.10. Fungerande system.

Efter en viss tid x, väljer kunden att serva en av slavarna och stänger därför ner den.



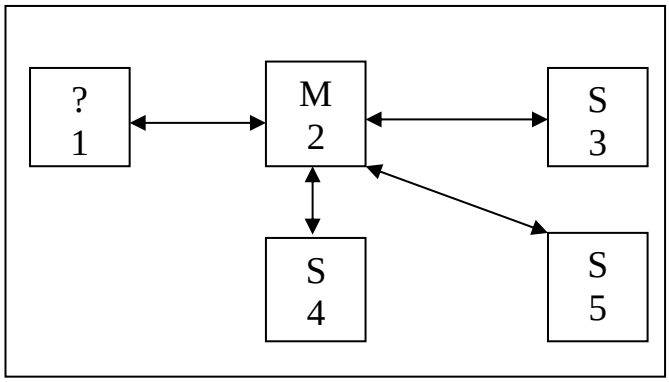
Figur 4.11. Slavenhet försvinner.

Mastern kommer då att se man får ingen signal tillbaka från en av slavarna, den tar den inaktiva slavens IP adress och raderar den från sitt register. Därefter skriver Mastern ut i sin display, att den tidigare aktiva slaven inte längre är aktiv. När detta är slutfört börjar Mastern igen att initiera sin kalkylering.
En annan fråga som dyker upp är slaven, bör den också radera den inaktiva drivenhetens IP adress från sitt register? Svaret kommer längre fram i senare kapitel.

4.5 New Driver

Vi har nu diskuterat scenarios där både en slav och en master försvinner, hur vi på bästa sätter åtgärdar dessa problem. Men då dyker nya frågor upp som vad händer om en ny drivenhet ansluts till vårt redan uppsatta och fungerande system? Är det en före detta slav i systemet där anslutningen avbröts? Eller har denna haft en egen anslutning till subnätet och trott sig vara den enda i systemet och på så sätt utsett sig själv till master? Vad skulle då hända om systemet inser att det nu kommit in ännu en master? Ett tänkbart scenario vi kommer även också att diskutera längre fram, är om just denna andra Master skulle ha en Slave under sig med.

Frågorna är en del och vi ska försöka svara och åtgärda all problematik som kan uppstå på bästa, lämpligaste sättet. Vi kan börja med att diskutera fallet där vår första drivenhet ej hittades under broadcastingens hitta/anslutnings fas och notera att vår första drivenhet har lägst IP-adress i subnätet, på så sätt vet vi att denna borde utsetts till master. Oturligt nog så hittas ej drivenheten ett och går därför över till drivenhet två som lyckades både hittas och anslutas, eftersom nu drivenhet två har näst lägsta IP-adressen och för tillfället minsta IP-adressen i vårt system, kommer just tvåan att utses till master.
Men vad skulle hända om systemet lyckas hitta drivenhet ett och ansluta den bland de andra enheterna i systemet?



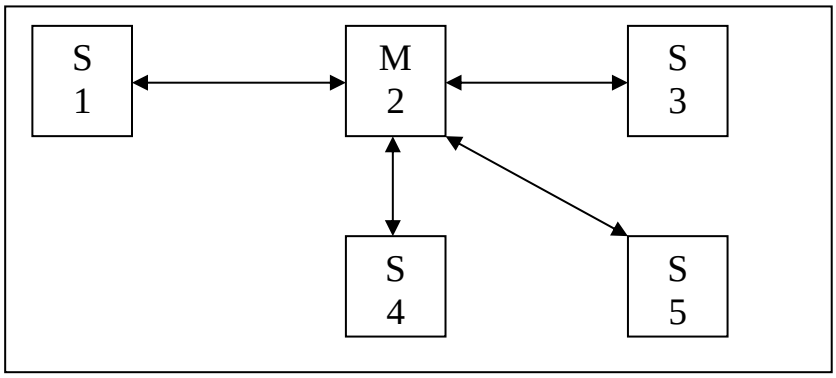
Figur 4.12. Detektering av en ny slav.

Eftersom drivenhet ett har lägst IP-adress och nu, efter lyckad broadcast, har hittats bestämde vi att den nyanslutna drivenheten kommer bli utsedd till en Slave, även om den i nuläget har lägst IP-adress. Varför vi väljer att inte byta master är på grund av att drivenhet två redan tagit rollen och jobbar kontinuerligt med att kontrollera de resterande slavarnas tillgänglighet, att deras givna ordrar följs, eventuellt dela ut nya om så krävs.

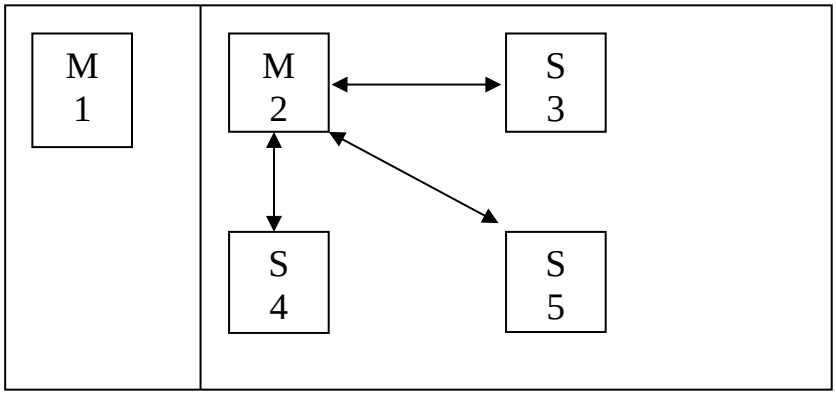
Om man nu skulle byta master hade detta först och främst krävt ett byte mellan den redan körande tvåan, initiera TFNM för att hitta den drivenheten med lägst IP-adress, som i vårt fall blir drivenhet ett. Inte nog med att TFNM

kan ge oss en allmän tidsfördröjning, utan ordrarna och de andra slavarnas tillgänglighetskontroll blir även försenade i värsta fall. När man väl äntligen utsett ettan till master, måste denna göra om ”broadcastingen” och pinga alla slavar, ansluta till dem för att meddela alla slavar om deras nya ordergivare.

Detta kommer leda till ytterligare en tidsfördröjning till och ännu en ytterligare tidsfördröjning igen, då vi även måste låta mastern få ta emot ”acknowledgments” av alla lyckade anslutna slavar innan den kan börja fördela arbetet och kontrollera respektive anslutning kontinuerligt. Därför väljer vi att drivenhet två fortsätter att köra som master och ettan blir en slav som dem andra och på så sätt besparar vi oss besväret.



Figur 4.13. Ny slavenhet ansluts till systemet.



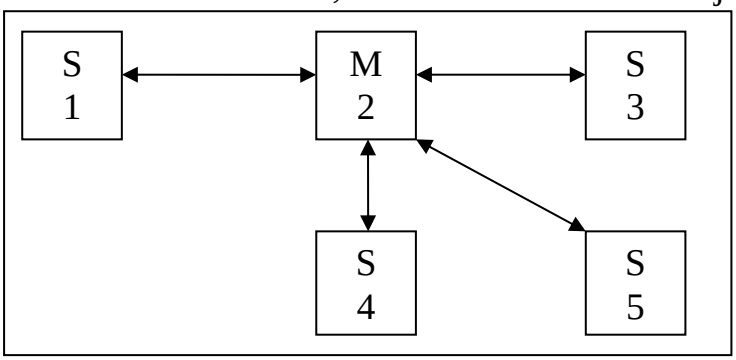
Figur 4.14. Två Masters i systemet.

Här ser vi i figur 2.07 att systemets broadcasting inte gick riktigt som vi hade tänkt oss och har delat in drivenheterna i två självständiga nätverk kan man säga. Eftersom i det ena fallet där ettan har lägst IP-adress och hittar endast sig själv under sin broadcasting, utser den direkt sig själv som master och den enda arbetaren i systemet. Medans på andra sidan, hittar man inte ettan och ger då tvåan master rollen(som nämnt tidigare så har tvåan näst lägsta IP-adressen) och man lyckades ansluta resterande drivenheter.

Låt oss nu säga att antigen ettan eller tvåan gör sin vanliga och regelbundna cykel uppdatering där den kollar att alla fortfarande är med i anslutningen,

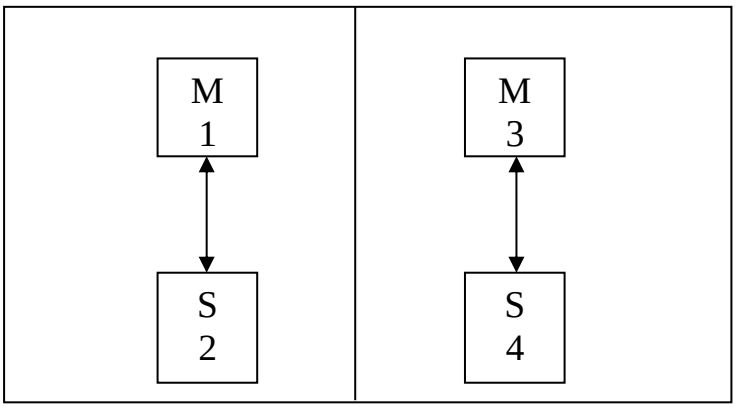
skulle på så sätt lyckas hitta ettan eller tvärtom, just då uppstår problematiken att det existerar två masters i vårt system. Vem av dem får stanna kvar som Master? Hur avgörs frågan?

Vi har sagt sedan tidigare att mastern bestämts efter lägst IP-adress, men så är ej fallet i detta läge heller. Eftersom tvåan har fler slavar än ettan så blir den enda mastern i systemet när ettan ansluts gemensamt till dem andra drivenheterna och ettans status ändras till slav och på så sätt slipper vi också onödiga tidsfördröjningar som kan uppstå om man hade valt att ettan ska få vara den enda mastern, detta nämndes mer detaljerat i föregående exempel.



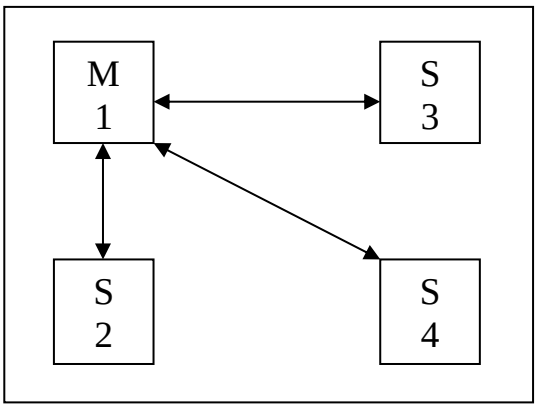
Figur 4.15 Master blir till slav.

Men om vi då har t.ex. jämt antal drivenheter och om samma fall upprepas igen?



Figur 4.16. Två Masters i systemet med lika många slavar under sig.

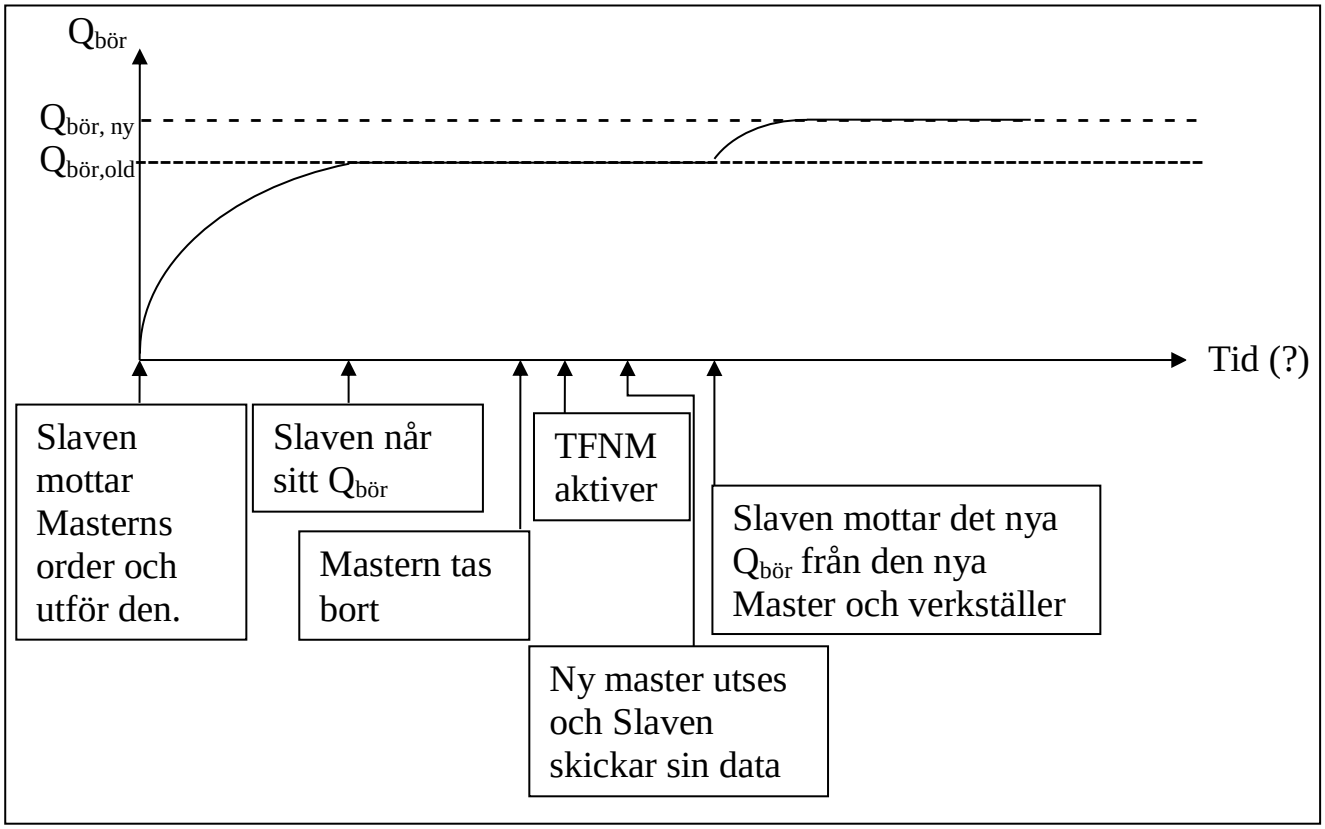
Här har båda masterna vars en slav under sig, om dessa två masterna skulle under någon cykeltid hitta varandra, måste en master dominera och i detta fall blir det som i dem vanliga fallen att drivenheten med lägst IP-adress tar över och därför blir ettan master, trean sänks till slav-status.



Figur 4.17. Masterenhet nummer tre blir slav.

4.6 Worst Case Scenario (WCS)

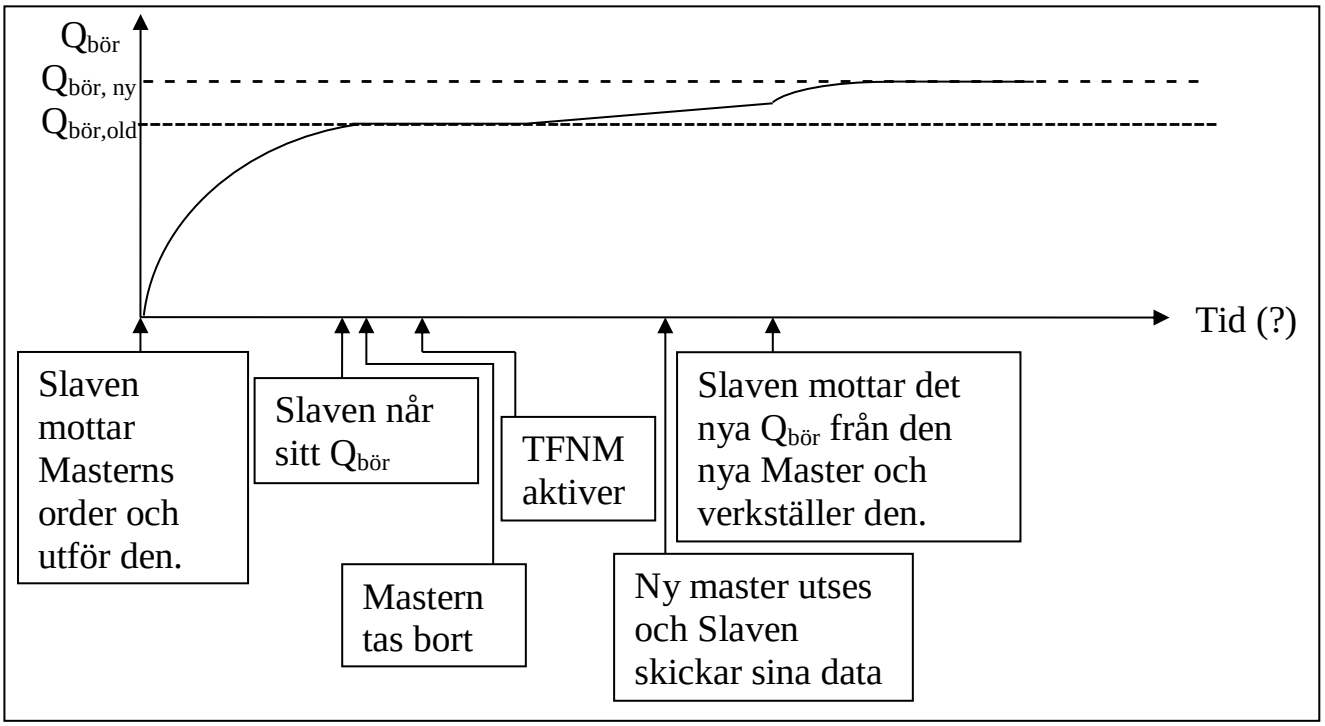
Om en Master försvinner och under tiden som Slaven väntar på en Master, vad händer med flödet, $Q_{\alpha,bör}$, för varje slav? Ska man låta den fortsätta pumpa ut samma flöde som den försvunna Mastern gav order om? Då kunden måste ha det till sina vattenskärare, som är beroende av flödet, måste man behålla samma flöde tills man har nått det nya flödet. Vi börjar med att undersöka Figur 2.18, OBS! Den innehåller inga data från tester, utan är ett exempel och används för att motivera vissa saker.



Figur 4.18a. Teoretisk bild för återhämtning av flödet.

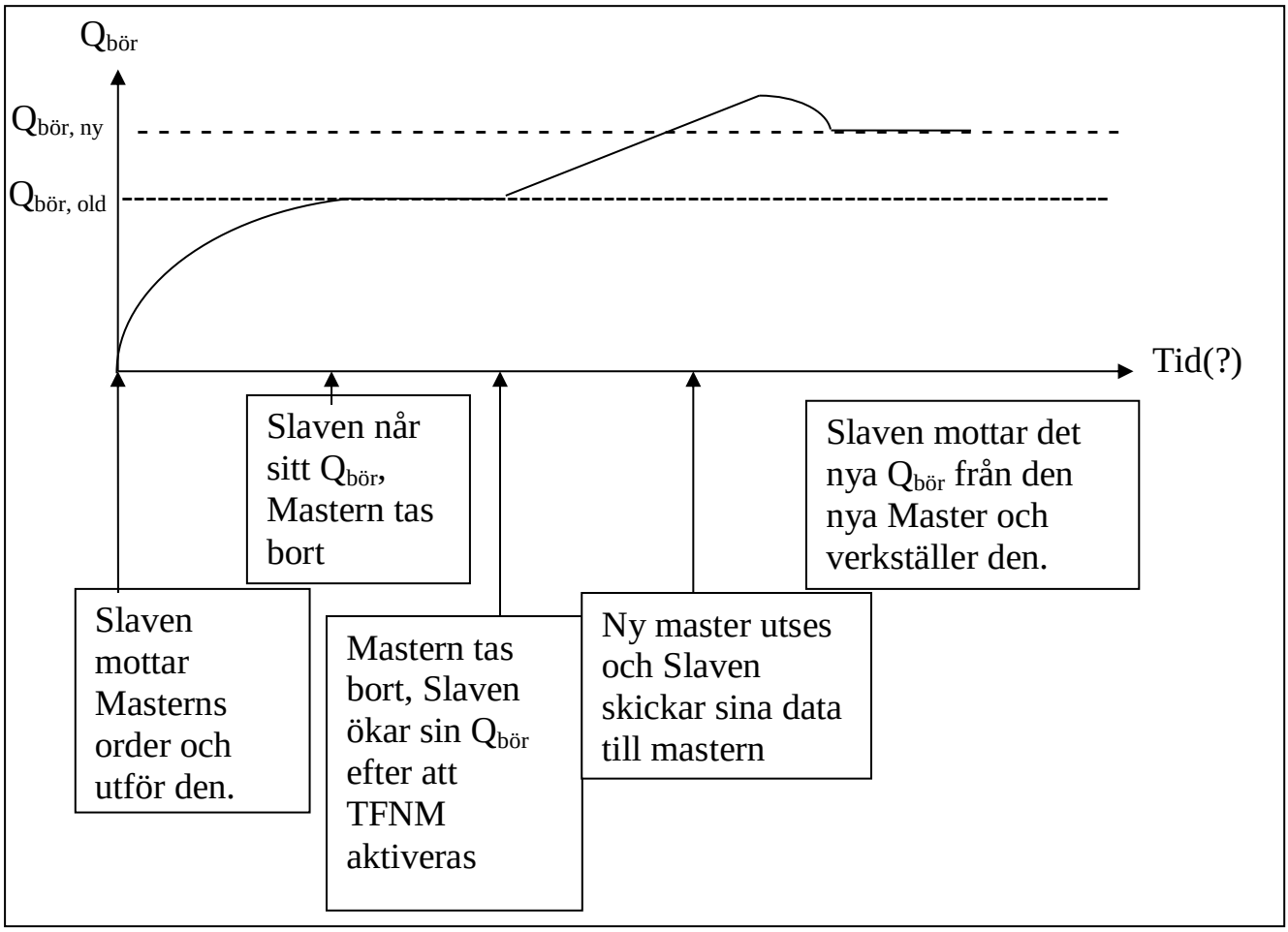
Om vi studerar figur 2.18, ser vi hur själva initial fasen sker samt hur flödet kommer att se ut före Mastern försvinner och efter att den nya Mastern har utsetts.

Som tidigare nämns, måste man undersöka hur lång tid det tar för en master att motta och skicka sina order till sina slavar, och därefter hur lång tid det för att verkställa den. Eftersom det **kanske** lönar sig att låta slaven öka sin $Q_{bör}$ medans systemet söker efter en ny Master.



Figur 4.18b. System återhämtning av flödet med linjäritet.

Notis: tiden för hur lång tid det tar för system återhämtning är i dagsläget okänt



Figur 4.19. System återhämtning av flödet med översväng.

Båda alternativ kan användas, men har olika förutsättningar. Alternativ ett är lämpligt om Master inte har någon arbetsbörda, medan alternativ två kan använda för att kompensera förlusten av en Master med en arbetsbörda. Problematiken i alternativ två, är hur ska systemet kompensera förlusten av en Master? Låter vi slavarna öka med ett procentuellt värde, blir risken att vi får en översväng i alla drivenheter.

Låt oss undersöka problematiken i alternativ två utan procentuell ökning. Vem ska bestämma? Hur stor ska flödet vara? Hur ska den bestämmas? Vilka villkor måste finnas? Vem ska kompensera för förlusten? När Mastern har utfört algoritmen, kommer den ha informationen om vilka drivenheter som ska nollställas och börvärde för varje drivenheten. Kan vi med den här sortens information lösa problemet?

Undersökningen visar att det går, men Mastern måste först låta huvudalgoritmen köras, sen måste den beräkna ett Worst Case Scenario (WCS) flödes värde med hjälp av parametrarna från algoritmen. Om Mastern står för den totala arbetsbördan i systemet, blir det problematisk som tidigare nämnt, då systemet blir beroende av den samt vet vi inte vet hur lång tid det tar för systemet att återhämta sig vid en eventuell förlust av en Master.

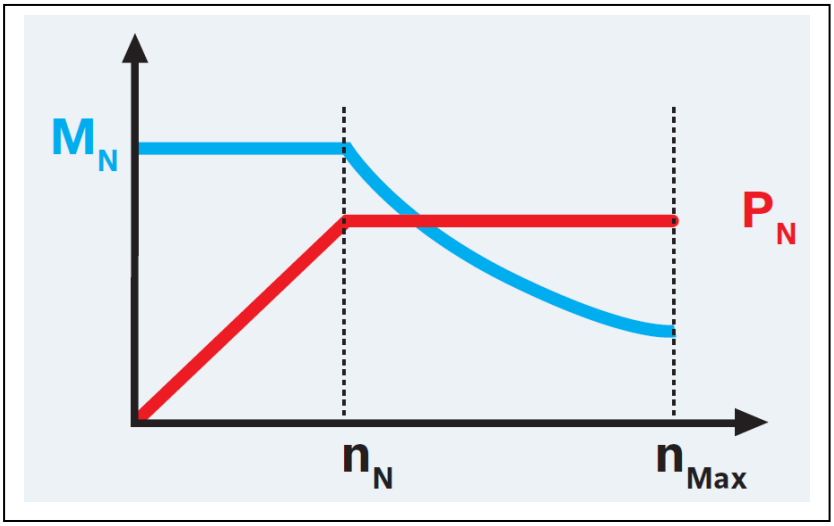
För att åtgärda detta problem, måste Mastern, innan den börjar skicka order, kontrollera vilken slav som kan kompensera med flödet Q_{WCS}, om Mastern försvinner. Värdet för flödet Q_{WCS} blir samma värde som börvärdet hos Mastern. Kontrollen utförs genom att kontrollera om det finns någon slav i systemet som kan kompensera förlusten. Villkoret för systemet blir, om slaven inte är i arbete och inte nått full kapacitet och $Q_{slav,max}$ större än eller lika med Q_{WCS}, sätts slavens Q_{WCS} till Masterns börvärde, och aktiveras endast om Mastern försvinner. Vi tar ett exempel för att underlätta.

Ex.
Notera i detta exempel har alla drivenheter samma maxkapacitet. Efter att algoritmen har körts, får Mastern informationen att den själv ska arbeta 100 % och Slaven nummer två ska också arbeta 100 %. När Mastern har kommit i detta läge att den har informationen om vilka som ska arbeta och icke arbeta. Mastern börjar undersöka vem som kan vara potentiell arvtagare av sin arbetsbörda. Den utesluter först alla som är redan i arbete,

Slaven nummer två i detta fall. Orsaken till detta blir att konsekvensen om vi passerar det nominella värdet, förlorar vi moment. Därefter måste den kontrollera om slaven med den minsta IP adressen kan hantera detta värde, villkoret blir om $Q_{slav,max}$ är större än eller lika med Q_{master}, stämmer detta, sätt slavens Q_{WCS} lika med $Q_{master, bör}$. Slaven sätter endast sitt börvärde till Q_{WCS} då Mastern har försvunnit, och arbetar med denna last framtill man har fått ett nytt börvärde av den nya Mastern.

Det tänkta systemet efter algoritmen	Worst Case Scenario (Mastern försvinner)
Q_1 = 100 % Q_2 = 100 % Q_3 = 0 % Q_4 = 0 % Q_5 = 0 %	Q_1 = 100 % Q_2 = 100 % Q_3 = 100 % Q_4 = 0 % Q_5 = 0 %

Ett annat problem som kan dyka upp om alla i systemet kör på sitt nominella värde, dvs. 100 %. Om det är så pass illa att ingen kan kompensera för förlusten, hur ska vi hantera problemet? Vi kan fortfarande fördela arbetet mellan resterande, konsekvensen av detta blir att vi förlorar moment, då vi passerar det nominella värdet.



Figur 4.20 Exempel på förlorat moment .

Det tänkta systemet efter algoritmen	Worst Case Scenario (Mastern försvinner)
Q_1 = 100 % Q_2 = 100 % Q_3 = 100 % Q_4 = 100 % Q_5 = 100 %	~~Q_1 = 100 %~~ Q_2 = 125 % Q_3 = 125 % Q_4 = 125 % Q_5 = 125 %

Värdet $Q_{\alpha,\,bör}$ och Q_{WCS} som Mastern har kalkylerat fram för varje drivenhet, skickas till respektive, där $Q_{\alpha,\,bör}$ används i Slaven för att få den att ställa in sig efter önskad värde. Medan Q_{WCS} sparas i Slaven och ställer in sig till sitt flöde endast om TFNM aktiveras, dvs. om Mastern försvinner.

Vi går igenom hur funktionen i WCS bör se ut. I beräkningen ignorerar man Mastern, det enda man räknar med är Slavarna. Det första som kontrolleras är att om det finns Slavar som är nollställda. Finns det inte det, använder vi WD funktionen och fördelar arbetet jämnt emellan Slavarna. Finns det? Då tar vi Slavens maximala flödes kapacitet. . Kan man kompensera förlusten? Är svaret nej? Addera det med Slaven därpå. Kan man kompensera förlusten? Är svaret ja? Dela arbetet emellan dem.

$Q_{\alpha+1,max} >= Q_{master,bör}$?
FALSE
$Q_{\alpha+1,max} + Q_{\alpha+2,max} >= Q_{master,bör}$?
TRUE

$$\frac{Q_{master,bör}}{Q_n} = Q_{WCS}$$

Skulle det finnas endast en nollställd drivenhet och den skulle inte räcka till för att kompensera förlusten måste man använda sig av en variant av OFC funktionen och därefter använda WD funktionen igen.

$Q_{\alpha+1,max} >= Q_{master,bör}$?
FALSE
INGA FLER

$$Q_{master,\ bö\ r} - Q_{\alpha+1,max} = Q_{master,bör}$$

$$\frac{Q_{master,bör\ ny}}{Q_n} = Q_{WCS}$$

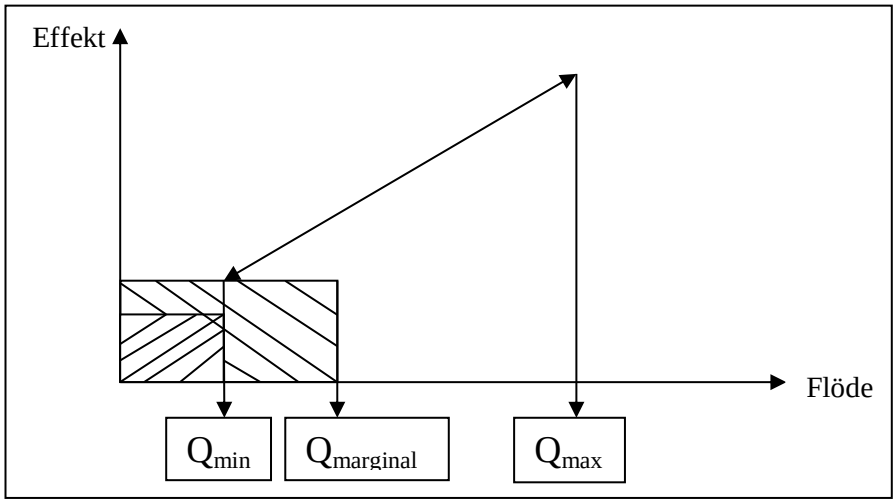
Detta värde sparas hos Slaven/Slavarna och sätts som börvärde endast om TFNM, dvs. att Mastern har försvunnit.
Vi får samtidigt inte glömma att flödet Q_{tot} är ett konstant kontinuerlig föränderlig värde. Det kan vara så att man inte behöver den här funktionen, eftersom det inte tar lång tid för att systemet ska återhämta sig från en eventuell förlust av Master. Funktionen har utvecklingsmöjligheter att spara tid vid eventuell förlust, frågan är dock hur mycket man sparar. Det enda möjliga sättet att svara på detta är om man utvecklar vidare denna funktion och simulerar detta.

4.7 Marginal

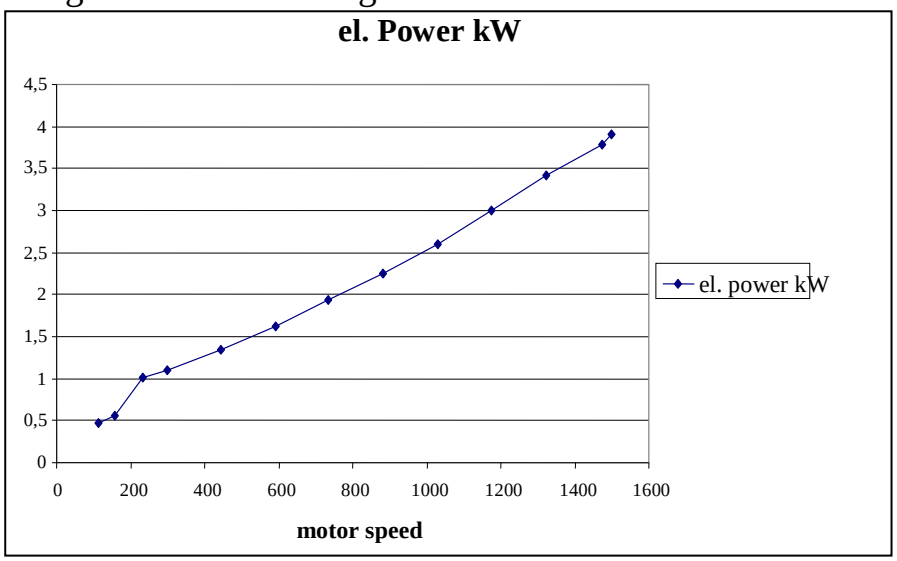
En funktion som undersöks är när en drivenhet bör ta över en annans arbetsbörda, samt vilka förutsättningar som gäller då. Ett av delmålen med systemet är att alla drivenheter ska vara potentiella Masters och Slaves, samt ska arbetsbördan fördelas jämnt. Det man måste tänka på är att funktonen inte får hindra systemet att nå en jämn arbetsfördelning när energibesparingen är optimalt.
Ett fall där funktionen kan tänkas använda är om förutsättningarna $Q_{\alpha,bör}$ är mindre än $Q_{\alpha,min}$. Under dessa omständigheter borde man minska en drivenhets flöde till $Q_{\alpha\ .bör}$ till noll och låta en annan drivenhet ta över arbetsbördan. Vilken drivenhet ska bära detta ansvar? Ska man välja beroende på motor storlek eller vilken rank en drivenhet har? Mastern är den som bär ansvaret för

att kontrollera att allt flyter på i systemet, men borde den ta över ens arbetsbörda? Vi börjar med att undersöka ett av villkoren för funktionen.

Låt oss säga att våra drivenheter får ett $Q_{bör}$ värde som är fortfarande ett värde som inte är energi ”vänligt” och inte under Q_{min} utan strax över det värdet. Hur löser vi problemet? Genom att införa $Q_{marginal}$, som omfattar ett visst område. Problemet som dyker upp med denna lösning är vilket värde ska $Q_{marginal}$ vara? Ska den vara 30 %, 40 % eller 50 % av Q_{max}? Värdet som $Q_{marginal}$ bör ha, måste undersökas.



Figur 4.21. Bild över gränsvärdena.



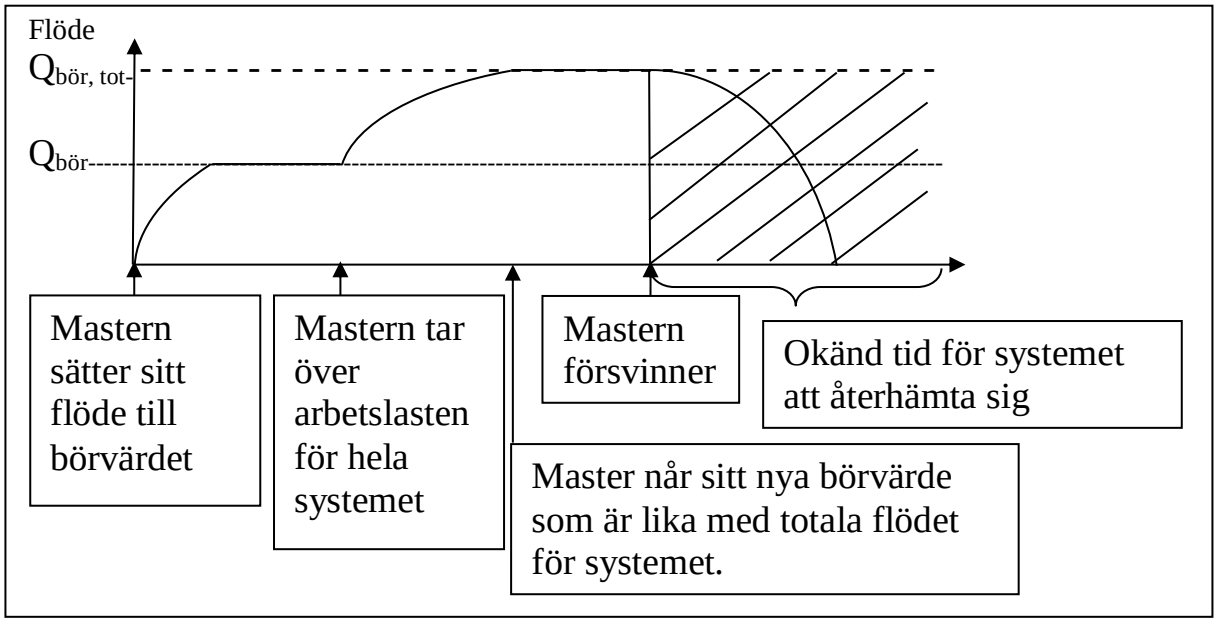
Figur 4.22 [6]. Bild över energikonsumption.

Om $Q_{\alpha\ .bör}$ är större än eller lika med $Q_{\alpha\ .min}$, vi kallar villkoret för Minimum Flow (MF), ska drivenhet$_\alpha$ minska sitt flöde till noll och låta en annan drivenhet ta över, i så fall vem? I algoritmen ska funktionen för arbetsfördelning, kapitel Work Divider (WD), fördela jämnt $Q_{\alpha,\ bör}$ till alla drivenheter. Men vad händer om WD:s beräkning ger ett börvärde som är

större än minimum värdet, samt så pass låg att det rent energimässigt inte passar? Den som borde ta över, är nämligen den drivenhet som har lägst motorstyrka. Om det skulle finnas motorer med samma storlek, tar man den drivenheten som har lägst IP, för på det sättet, kan systemet automatiskt välja vem som ska ta över arbetslasten, beroende på vilket max flöde värde en drivenhet har, vi löser även problematiken om vem av drivenheterna ska väljas om flera motorer har samma poler.

Dock är det viktigt att notera, Mastern bör **inte** ensamt stå för arbetet i systemet, då vi i dagsläget inte vet hur lång tid det tar för systemet att välja ny Master, kalkylera och få systemet att köra normalt igen. Man kan dock ändå göra detta, men då måste man kompensera detta vid en eventuell förlust av en Master, fallet Worst Case Scenario beskriver detta. Undersökningen visar, med hjälp av fall Slave Flow With No Master, att detta inte är lämpligt. Om detta sker och all arbetsbörda hamnar hos Mastern, och efter en viss tid försvinner den, kommer all flöde att försvinna. Kunden som är beroende av ett visst flöde kommer vara tvungen att stoppa sin produktion tills en ny Master har utsetts i systemet. Det är viktigt att notera, denna konklusion använder vi eftersom vi inte vet hur lång tid det tar för systemet att återhämta sig. Bilden nedan visar en dramatisk bild för om Mastern tillåts ta över arbetslasten för systemet och sedan försvinner (Figur 4.22).

Vi väljer inte att ta med Slavens responstid, då vi inte vet detta i dagsläget. Tidigare i texten nämns det att Mastern är den viktigaste rollen, då det är den som kontrollerar, kalkylerar, skickar och tar emot data. Slavarna är ”beroende” av Mastern då den förväntar sig att få information om att Mastern är där, ta emot order och skicka data till den. Skulle Mastern tillåtas ta över hela arbetsbördan, skapar man ett beroende system istället för ett oberoende, vilket gör att vi missar ett av delmålen med systemet. Slavar är dock också oberoende av Mastern, eftersom om Mastern faller bort, kommer Slavarna att själva utse en ny Master individuellt.



Figur 4.23. System återhämtning

Om en Slave tar över den totala arbetsbördan däremot, istället för en Master, undviker vi att skapa ett system som är beroende av en Master, om Mastern får stå för hela arbetsbördan i systemet. Eftersom om en Slave väljs till att utföra all arbete, för att sedan försvinna, kommer Mastern att detektera detta och antingen fördela arbetet eller låta någon annan Slave ta över, beroende då på vilka villkor som uppfylls.

Men samtidigt betyder inte detta att man når alltid en energioptimering av systemet. Det kan hända att man egentligen bör köra endast Mastern då detta är mer energi effektivt, då måste slavarna kompensera denna tänkbara förlust när Mastern har försvunnit. Vi diskuterar detta längre fram i texten.

Vi har nämnt tidigare att man kan använda sig en marginal, men inte riktigt nämnt behovet av den. Bakgrundshistorien ligger i att om man inte kör i sitt nominella varvtal och överdimensionerar, får man förluster, vilket inte är praktiskt i energi sammanhang. Om vi införde ett marginal värde som berättar för oss i vilket intervall det är icke rekommenderat att köra en maskin energimässigt, kan vi på så sätt undvika överdimensionering. Ett av problemen här, blir som tidigare nämnts att vi vet inte vilket värde marginalen ska ha, ska den vara på 30 % av maximal kapacitet? Detta är ett problem som löses via simuleringar, där man jämför värden och hittar lämpligt värde för marginalen. Tanken här är att om $Q_{\alpha,\ bör}$ är under i värdet $Q_{marginal}$, ska systemet tolka detta som att finns möjligheten att nollställa drivrarna, vilket aktiverar funktionen

LFC. Vi får dock inte glömma bort, som tidigare nämnts i texten, att användaren kan av okänd anledning eftersträva ett lågt börvärde, som ligger i under värdet $Q_{marginal,}$. För att lösa problematiken måste vi ha villkor för detta. Men samtidigt måste vi också ha ett villkor för vem som ska ta hand om arbetslasten. Villkoret för detta blir, om det totala eftersträvade flödet är mindre är marginalflödet **och** det totala flödet är större än det minimala kapacitets flöde, är detta ett tecken på att användaren eftersträvar ett flöde som är under det rekommenderade intervallet.

Stämmer detta villkoret, måste man ta åtgärder. Men vilka åtgärder ska ske? Rent logiskt kan man med ett sådant lågt värde, tillåta endast en maskin att arbeta. Funktionen måste dock fortfarande kontrollera att man inte är under drivenehetens minimala kapacitet. Om det är det, då övergår man till nästa drivenhet och kontrollerar.
Själva funktionens uppbyggnad blir, är om det börvärdet $Q_{bör}$ mindre än eller lika med $Q_{marginal}$ **och** är $Q_{bör}$ större än eller like med Q_{min}. Stämmer villkoret, aktiveras funktionen och börjar med att kontrollera drivenheten med den lägsta IP adressen om den kan hantera ett så pass lågt värde, kan den inte, övergår den till nästa drivenhet i registret.
Men vilken värde på flödet $Q_{marginal}$ ska man ha? Ska man titta på drivenheten med den största maximala kapaciteten i systemet (om det finns sådan), eller drivenheten med den minsta maximala kapaciteten (om det finns sådan). Vi börjar med undersökningen av detta.
Om vi har fem drivenheter, varav drivenheternas maximala kapaciteten är ekvivalenta med varandra, förutom drivenheten med flödet Q_1, den har största maximala kapaciteten i systemet. Ska man ta $Q_{marginal}$ från den största eller de minsta? Man bör använda sig av det största marginalvärde som finns i systemet, eftersom att ge en större motor med en låg arbetslast kan bidra med förluster, om överdimensionering uppstår. Om drivern med den maximala kapacitetens marginal värde är större än det eftersträvade flödet i systemet, $Q_{bör}$, kan man säkerställa att användaren vill ha ett extremt lågt värde. Dock måste man kontrollera om drivenheten kan hantera detta låga värde, kan den inte, kontrollerar man nästa drivenhet, villkoret för detta blir:

$$Q_{\alpha,min} \leq Q_{\alpha,bör}$$

Vi repeterar igen hur funktionen bör se ut. Det första som görs i funktionen, är att man kontrollerar om det finns en drivenhet med en minimal kapacitet som är mindre än de andra drivenheterna, om det finns två eller flera sådana, använder man drivenheten med den lägsta IP adressen, man gör en IP prioritering med lägsta adress. Finns det ingen sådan, tar man drivenheten med lägsta IP adressen i systemet och använder sitt $Q_{marginal}$. Därefter kontrollerar man om den drivenheten kan hantera detta låga totalvärde för flödet, om inte, kontrollerar man nästa drivenhet, osv.

4.8 Algoritm

Algoritmen är både den centrala delen och delen som ska stå för vårt systems drift. Den kommer att stå för det största ansvaret genom att kalkylera och hålla koll på data som finns i varje drivenhet, inklusive mastern. Eftersom alla drivenheter ska kunna vara master och leda hela systemets arbetsprocess, kommer alla drivenheter att inneha algoritmen. Algoritmen ser olika ut hos drivrarna, beroende på om det är en slav eller master och det är baserat på algoritmens kalkyleringar och beslut som mastern kommer att fördela arbetet till slavarna, med hänsyn till det mest effektivaste sättet.
Hur gör vi då för att uppnå det effektivaste sättet och hur fördelar Mastern allt? Vad behöver Mastern respektive Slaven veta om varandra? Vilka data är viktiga att veta och vilka data faktorer är viktiga att skicka?
Dessa frågor kommer vi bland annat att behandla i följande textsnitt.

Vad mastern behöver ha för egenskaper och kännedom:

- Antalet slavar den kan ge order till.
- Varje slavs data respektive utdata.
- Kunna detektera om en slav försvinner och på så sätt försöka återansluta denna.

Vad slaven behöver ha för egenskaper och kännedom:

- Nuvarande mastern.
- Kunna skicka och ta emot data till mastern och verkställa ordarna från mastern.
- Kunna detektera om mastern försvinner och kolla genom registret efter nästminsta IP-adressen

Bland de viktigaste data vi kommer att vara intresserade över är $Q_{max,}$ $Q_{bör,}$ $Q_{min,}$ där Q står för flödet. Maximala flödet för en drivenhet, $Q_{max} = n_{max} * V * \eta$, där n_{max} är drivenhet motors varvtal multiplicerat med deplacementet som är volymen i vår pump (V), angivet i kubikcentimeter, multiplicerat med verkningsgraden (0.9). Som vi nämnde innan, kommer användaren att mata in önskat maximal och minimum flödet Q_{max} och Q_{min}. $Q_{bör}$ är börvärdet för en drivenhet och kan beräknas på olika sätt beroende på motorernas storlek är lika eller olika, vilket ställer kravet att systemet borde göra en jämn respektive

ojämn arbetsfördelning med hänsyn till varje motors nuvarande arbetskraft och kapacitet.
Hur $Q_{bör}$ beräknas kommer vi att ta upp längre fram, men något annat viktigt att ta hänsyn till är Q_{min} som är vår minimala flödesgräns innan det lönar sig med att stänga av en motor, för att minska effektförlusterna. Men något som är värt att också nämna är $Q_{marginal}$ som är en gräns efter Q_{min}, $Q_{marginal}$ gränsen ligger tät intill $Q_{min,}$ om flödet skulle hamna vid denna gräns skulle motorn fortfarande få tillåtas att köras, dock orekommenderat från ett energisnålt perspektiv. Eftersom flödet är beroende av motorns varvtal, kommer vi att behöva varvtalsreglera, medans den variabla pumpens flöde regleras med hjälp av en vinkel α.
Eftersom vi kör med både varvtalsreglering med våra motorer och med variabla pumpar, kommer detta ge oss ett dynamiskt börvärde i systemet, $Q_{bör}$. Det är viktigt att flödet hela tiden anpassar sig både för en ökning och nedsänkning för att tillfredsställa kravet på det önskade trycket, för att inte på så sätt avbryta, t.ex. vattenskärarmaskinens arbetsprocess.

4.9 Struktur för Algoritmen

Algoritmen bör struktureras på en sådan sätt att den förhindrar detta. Men hur ska den struktureras? Den måste även struktureras så att den inte tillför alltför stora tidsförseningar i systemet. Vi har i våra undersökningar hittat tre funktioner som har fyra olika syften i algoritmen.

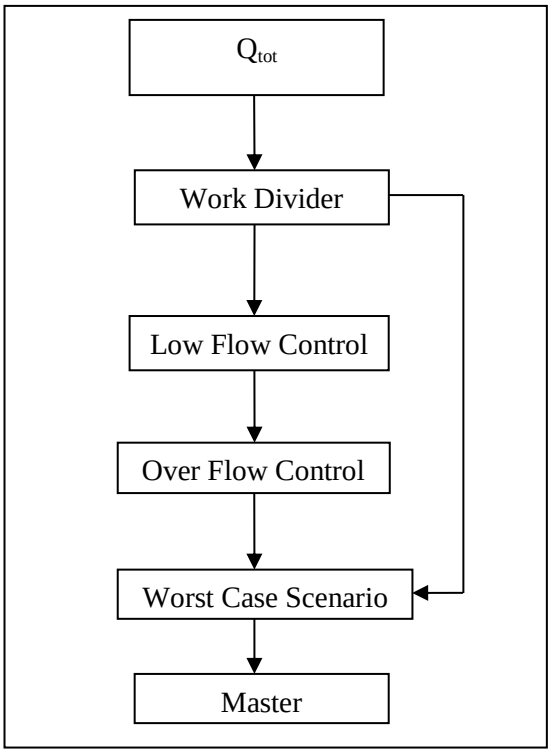
1. Funktionen WD (Work Divider), fördelar arbetet mellan drivenheterna.
2. Funktionen LFC (Low Flow Control), kontrollerar om en eller flera drivenhet är under sitt marginalvärde för flödet, samt om när det är lämpligt att nollställa en drivenhet.
3. Funktionen OFC (Over Flow Control), kontrollerar om en drivenhet eller flera drivenheter är över sin maxkapacitet och åtgärdar felet.
4. Funktionen WCS (Worst Case Scenario), beräknar fram ett börvärde som är tänkt att Slaven ska spara och användas för att kompensera flödesförlusten i systemet, ifall Mastern försvinner.

Hur ska algoritmen struktureras upp med dessa funktioner och vilka ska ske först? Den första funktionen som ska ske, är WD funktionen, resultatet ska sedan skickas vidare till nästa instans, eftersom här kan vi se om det tänkta börvärdet är alltför låg eller hög för en drivenhet.
Ett av problemen som kan uppstå i systemet efter WD funktionen, är att man får ett börvärde som är under en drivenhets marginalvärde och samtidigt i en

annan drivenhet överskrids den sin maximala kapaciteten, vilket av problemen ska lösas först? Vi väljer att åtgärda problemet med att börvärdet understigs först.

Motiveringen till detta är att om vi understiger marginal/minimum värdet, säger detta oss att det är lämpligt att nollställa drivenhet/ drivenheterna, vilket gör att vi sparar energi då vi inte behöver ha alla drivenheter i arbete. Om vi byter prioritering får vi en större tidsfördröjning i systemet, eftersom det kan hända att den drivenhet som först kör igenom OFC ska sättas till en 100 %, medan efteråt visar det sig att man egentligen behöver stänga av den efter man kört LFC funktionen.
Därefter kanske man får ett värde som överskrider den maximala kapaciteten i LFC, då måste man skicka detta till OFC funktionen igen för att kalkyleras igen, därefter kan Mastern och Slavarna arbeta, vilket de kunde ha gjort tidigare om det inte var för det ytterligare tidsfördröjning som uppstod när vi bytte prioritering. Själva tanken var detta med prioriteringen, men vid närmare undersökning av LCF funktionen, visar det sig att OFC funktionen inte egentligen behövs, motiveringen till detta kommer längre fram i.

Tänkbar struktur för algoritmen:



Figur 4.24

WD funktionen får ett kontinuerligt flödes värde för det önskade börvärdet (flödet), $Q_{bör}$, i systemet, därefter beräknar fram det önskade börvärdet för systemet, därefter skickas denna information till LFC eller direkt till WCS beroende vilket val maskinoperatören har valt, där kontrollerar/ beräknar den fram vilka drivenheter som behöver stängas och vilket nytt börvärde det ska vara. LFC skickar vidare informationen om vilka drivenheter som ska vara med i framtida beräkningar, samt det nya börvärdet, vidare till OFC.

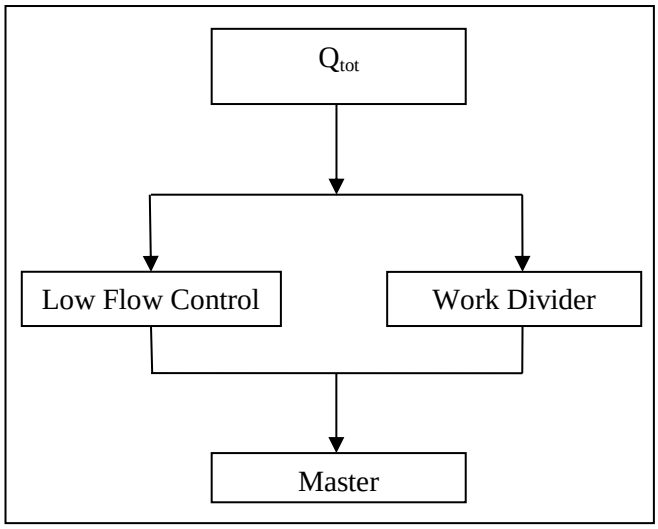
Funktionen OFC ignorerar de drivenheter som inte var med i datan som skickades av LFC och kontrollerar endast de som ska vara med, om det visar sig att en drivenheterna börvärde överskrider ens maximala kapacitet, beräknas ett nytt börvärde. OFC:s beräkning ger Mastern svar på vilket börvärde varje drivenhet ska arbeta med, men också vilka som ska nollställas. Denna information skickas till WCS funktionen. Den beräknar fram ett värde för att kompensera vid en eventuell förlust av en master. Denna information skickas med data från OFC. Data som OFC skickar till WCS, måste innehålla vilka börvärden respektive drivenhet ska ha och vilka som ska nollställas.

Men om vi undersöker LFC funktionen så ser vi att man egentligen inte behöver OFC, då LFC löser i stort sett problemet, då tanken är att den arbetar på ett sådant sätt som gör att vi inte behöver kontrollera om ens börvärde överstiger ens drivenhet maximala kapacitet.

WCS funktionen behövs kanske inte om systemet är snabb med att ersätta en Master. Dagens Slavar kommer att börja arbeta oberoende varandra om de mister sin Master och kompensera förlusten. Men om ersättningen av Master sker snabbt, blir WCS funktionen betydelselös.

Man kan säga att WCS en idé där man kan spara tid på att kompensera flödesförlusten när Mastern försvinner, genom att låta systemet beräkna detta.

Strukturen för algoritmen:



Figur 4.25

Den här strukturen som vi beskriver här är den designen vi har tänkt använda oss av i algoritmen. Om vi jämför med den tidigare strukturen, ser vi att vi har ”optimerat” den. OFC och WCS förkastas då de inte behövs, motiveringarna finns i föregående text. Tanken är dock nästan detsamma, man mottar ett dynamisk börvärde för systemet, därefter kan man gå två vägar beroende på vilken optimering maskinoperatören strävar efter, antingen vidare till Low Flow Control eller WD funktionen.

4.10 Algoritm för Automatisk Flödes Reducering/Ökning

Vi undersöker båda fallen där systemets arbetsbörda får vara beroende enbart av Master eller helt beroende. Först behandlas fallet oberoende

4.10.1 Work Divider (WD)

Hur ska då algoritmen beräkna fördelningarna och vilka åtgärder måste vi ta för att ändra i denna fördelning om flödet skulle bli över någons Q_{max}?
När vi beräknar en arbetsfördelning bland drivenheter får vi ut ett flödesbörvärde för respektive drivenhet, detta börvärdet, $Q_{bör}$,beräknas som nämnt tidigare, totala flödet i systemet dividerat med antalet anslutna drivenheter, det vill säga, $Q_{bör} = \frac{Qtot}{Drivers}$, denna beräkning kallar vi för ”work divider” och vi kommer att referera till denna beräkning som ”WD” i detta examensarbete.

Om det skulle hända t.ex. att drivenhet två har ett börvärde som är större än drivenhetens maxkapacitet, så att ($Q_{bör} > Q_{max}$), betyder detta att drivenhet två

håller på att ta emot en överbelastning. Hur vi löser detta problemet kan variera beroende på situationen och hur stor överbelastningen är. Vi tar även hänsyn till dem andra drivenheternas maxkapacitet, nuvarande arbetsbörda. Beroende på vad dessa är kan vi tillämpa jämn respektive ojämn arbetsfördelning. Eftersom vår master i systemet har en cykel tid på en millisekund, kommer den att märka av denna överbelastning. Direkt när mastern meddelats att drivenhet två i vårt fall har fått en överbelastning, kommer mastern att beräkna exakta värdet i liter per minut som överstiger maxkapaciteten. Sedan kommer mastern att kolla genom slav drivenheterna som har möjlighet att kunna ta emot överbelastningen från drivenhet två, och tilldelar antigen en motor eller flera jämnt eller ojämnt som vi nämnde ovan, detta görs med olika kända metoder som vi kommer ta upp senare.
Skulle överbelastningen för drivenhet två vara en liten del över bara, så skulle kanske den enklaste lösningen vara att använda ”quick-fix” som letar snabbt efter närmast och bäst drivenhet den hittar, som har möjligheten att kunna ta emot den här lilla arbetsbördan utan att $Q_{bör}$ för denna drivenhet kommer över dennas Q_{max}. Sökningen för en drivenhet som kan ta emot denna överbelastning görs av mastern, genom att den kollar i sitt register med alla slavar, från lägst IP-adress och uppåt.

När en lämplig ersättare hittas, skickar mastern då dels order till drivenhet två och dels till den drivenhet som ska ta emot arbetsbelastningen (det kan förekomma att mastern får ta emot arbetsbelastningen), dessa två slavar måste sedan skicka tillbaka en ”acknowledgement”, för att mastern ska veta att ordern har mottagits och håller på att verkställas. För att vara säker på att dem nya ordrarna verkligen verkställs, kommer master efter en viss cykel tid tagit emot data från alla slavar, där nuvarande arbetsbelastning rapporteras och även användaren kommer att kunna se skillnaden på sin panel.

Hade däremot arbetsbelastningen varit för stor för att en enda drivenhet ska kunna ta emot en sådan överbelastning, beräknar vi överbelastningen för drivenhet två och sedan låter man drivenheten köra på sitt Q_{max}. Den sedan subtraherade överbelastningen kommer man att göra WD på och tilldela det nya arbetssnittet $Q_{bör}$ genom att master ger ordern till resterade slavarna som förklarat ovan, på så sätt kommer slavarna att tilldelas lika stort arbetssnitt, om såklart alla kan klara av den nya arbetskraften, annars initieras WD igen.

Låt oss nu demonstrera ett praktiskt exempel på jämn fördelning med WD:

$Q_{bör} = 75$ l/min

drivenheterna D1-D4 har ett Q_{Max} på 20 l/min medan D5 har ett Q_{Max} på 10 l/min

Mastern börjar initiera WD, dvs. $Q_{Bör} = \frac{Qtot}{D}$, där D är antalet drivenheter och i vårt fall har vi fem drivenheter.

$Q_{\alpha,\,bör} = \frac{75}{5} \Rightarrow Q_{\alpha,\,bör} = 15$ l/min

$Q_{1,max} = 20$	$Q_{1,bör} = 15$
$Q_{2,max} = 20$	$Q_{2,bör} = 15$
$Q_{3,max} = 20$	$Q_{3,bör} = 15$
$Q_{4,max} = 20$	$Q_{4,bör} = 15$
$Q_{5,max} = 10$	$Q_{5,bör} = 15$

Om vi lägger märke till Q_5, ser vi att den inte kommer att uppnå det givna börvärdet $Q_{5,bör}$ eftersom börvärdet är större än drivenhetens maximala flödeskapacitet. För att åtgärda denna sortens problematik måste vi först låta drivenhet Q_5 köra på maxkapacitans och sedan utesluta den från systemets totala flöde Q_{tot}, genom att beräkna ut en ny Q_{tot} som blir:

$Q_{ny,\,bör} = Q_{bör} - Q_{5,max} = 75 - 10 \Rightarrow Q_{ny,bör} = 65$ l/min

När vi väl fått vår nya $Q_{bör}$ gör vi en ny WD beräkning som blir:

$Q_{\alpha,\,bör} = \frac{Qny,bör}{d-1} = > Q_{\alpha,\,bör} = \frac{65}{4} = 16.25$ l/min

Det nya $Q_{bör}$ verkställs av mastern och ökar drivenheterna D1-D4 arbetssnitt medans D5 förblir körande på sin $Q_{5,max}$.

$Q_{1,max} = 20$	$Q_{1,bör} = 16.25$
$Q_{2,max} = 20$	$Q_{2,bör} = 16.25$
$Q_{3,max} = 20$	$Q_{3,bör} = 16.25$
$Q_{4,max} = 20$	$Q_{4,bör} = 16.25$
$Q_{5,max} = 10$	$Q_{5,bör} = 10$

Vi kan summera $Q_{1\text{-}5,bör}$ och se att resultatet blir 75 l/min (16.25+16.25+16.25+16.25+10 = 75 l/min), som är det $Q_{bör}$ som man eftersträvar i början av processen.
I ett annat fall kan det löna sig att använda den andra metoden vi nämnde som vi kallar för ”quick fix”, vilka situationer exakt kommer vi att återkommer med längre fram. Men låt oss säga att vårt börvärde flöde, $Q_{bör} = 105$ l/min, vilket ger oss ett $Q_{\alpha,\,bör}$:

$Q_{\alpha,\,bör} = \frac{105}{5} = 21$ l/min

$Q_{1,max} = 25$	$Q_{1,bör} = 21$
$Q_{2,max} = 25$	$Q_{2,bör} = 21$
$Q_{3,max} = 25$	$Q_{3,bör} = 21$
$Q_{4,max} = 25$	$Q_{4,bör} = 21$
$Q_{5,max} = 20$	$Q_{5,bör} = 21$

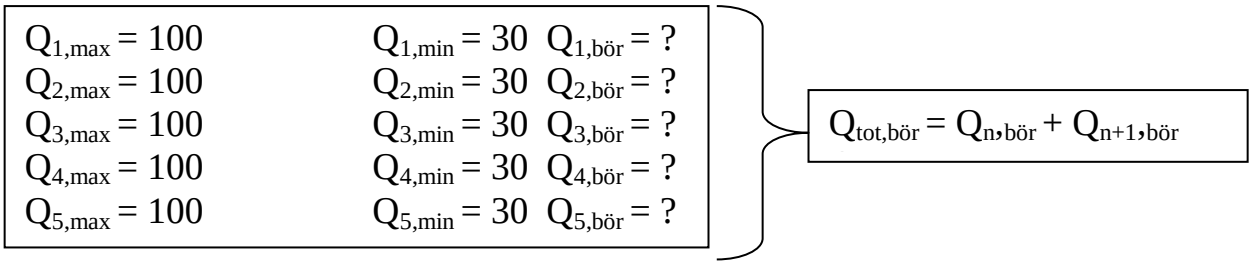
Tar vi hänsyn till alla drivenheters $Q_{max,}$ ser vi att Q_5 har fått lite arbete över sin kapacitet och med att initiera vår ”quick fix” som kommer söka efter snabbast och lämpligaste drivenheten att ta emot Q_5 lilla överbelastning, denna drivenhet kan lika gärna vara mastern själv som tar emot efter att den upptäckt överbelastningen genom sin cykeltid. Själva sökningen görs genom registret som innehåller alla IP-adresser, med att börja med lägsta IP-adressen och uppåt tills ersättaren är hittad. Låt oss säga att mastern som är Q_1 i detta fall tog emot överbelastningen från Q_5, då hade systemets arbetsprocess blivit stabilt och problemet löst med vår ”quick fix ” funktion.

$Q_{1,max} = 25$	$Q_{1,bör} = 22$
$Q_{2,max} = 25$	$Q_{2,bör} = 21$
$Q_{3,max} = 25$	$Q_{3,bör} = 21$
$Q_{4,max} = 25$	$Q_{4,bör} = 21$
$Q_{5,max} = 20$	$Q_{5,bör} = 20$

4.10.2 Selektiv Test

Funktionen borde struktureras på följande sätt. Mastern utför arbetsfördelningen, innan den skickar ett flödes börvärde, kontrollerar man om den/de mottagande drivenheterna kan utföra detta. Om det visar sig att flödes börvärde är under sitt flödes minimumvärde för drivenheten, påbörjar vi sökningen efter en Slave som med högst Q_{max}. Därefter kontrollerar vi om den kan hantera det, kan den det, sätter vi den till att arbeta i 100 %. Om inte den kan hantera detta, ökar vi antalet med Slavarna med ett och kör Work Divider funktionen igen. Mastern kontrollerar igen om de kan hantera detta, kan de hantera detta, skickar Mastern ut ordern.

För att förklara detta bättre tar vi två exempel.



Mastern har sedan tidigare information om den $Q_{\alpha,max}$ och $Q_{\alpha,min}$, som är 100 respektive 30 l/min i vårt exempel. $Q_{\alpha,marginal}$, har värdet 0.5 * $Q_{\alpha,max}$ (50% av värdet på $Q_{\alpha,max}$). $Q_{\alpha,bör}$ är inte fastställd än. Komponenten (vattenskärare) kräver ett flöde av systemet, $Q_{bör}$, som är 240 l/min. WD aktiveras och får att värdet $Q_{\alpha,\ bör}$ till 48 l/min. Detta värde är under $Q_{marginal}$, Mastern detekterar detta och börjar utföra sin kalkylering.

Funktionen tar drivenheten med den högsta Q_{max}, i vårt fall har alla lika stor Q_{max}, då ska man ta den drivenheten med minsta IP, nämns i början av fallet. Den kontrollerar om drivenheten med sin max värde, $Q_{2,max}$, kan utför det önskade arbetet av operatören. Kontrollen visar att detta inte kan ske och säger att villkoret för drivenhetens maximala värde inte är större än eller lika med det önskade $Q_{bör}$.

Mastern tar nästa Slav, Q_3, med högsta flödes kapacitet och lägsta IP:n. Är Q_2 och Q_3 maximala kapacitet tillsammans större än eller lika med det önskade flödet för systemet? Är svaret nej, lägger Mastern ytterligare till en slav med högsta flödes kapacitet och lägsta IP:n. Är svaret större än eller lika med det önskade flödet för systemet? Ja? Gå vidare till nästa steg.

$Q_{2,max}$>=240?
FALSE
$Q_{2,max}+Q_{3,max}$>=240?
FALSE
$Q_{2,max}+Q_{3,max}+Q_{4,max}$>=240?
TRUE
$Q_{2,max}+Q_{3,max}+Q_{4,max} = 300$

Vi låter funktionen även undersöka om vi kan låta en drivenhet nollställas, dvs. ger 0 liter/min. Programmet tar den drivenhet med den minsta maximala arbetskapaciteten. I detta exempel har alla samma maximala arbetskapacitet, vilket gör att funktionen tar drivenheten med minsta IP adressen. Och subtraherar med de drivenheter som inte valts ut. Kan Q_2 och Q_3 ensamma ta

hand om arbetslasten? Nej? Återgå till föregående värde då värdet var större eller lika med det önskade.

I detta exempel är det onödigt, men motiveringen kring varför man ska använda detta kommer i exempel två.

Ex 1:

$Q_{3,max}+Q_{4,max}+Q_{2max}>=240$?
TRUE
$Q_{3,max} + Q_{4,max} >= 240$?
FALSE

$$Q_{\alpha,\,bör} = \frac{240}{3} = 80 \text{ l/min}$$

När funktionen är klar med beräkningen ska de drivenheter som var med i beräkningen få sitt börvärde, som sätts till 80 l/min i drivenheterna Q_2,Q_3 och Q_4. De drivenheter som inte får ett uträknat börvärde ska nollställas.

$Q_{1,max} = 100$	$Q_{1,min} = 30$	$Q_{1,bör} = 0$ l/min
$Q_{2,max} = 100$	$Q_{2,min} = 30$	$Q_{2,bör} = 80$ l/min
$Q_{3,max} = 100$	$Q_{3,min} = 30$	$Q_{3,bör} = 80$ l/min
$Q_{4,max} = 100$	$Q_{4,min} = 30$	$Q_{4,bör} = 80$ l/min
$Q_{5,max} = 100$	$Q_{5,min} = 30$	$Q_{5,bör} = 0$ l/min

$$Q_{tot,bör} = Q_{n,bör} + Q_{n+1,bör}$$

Men vad händer om vi har drivenheter med olika värden på maximal/minimal kapacitet? Vi tar nu exempel två.

Ex 2:

$Q_{1,max} = 100$	$Q_{1,min} = 30$	$Q_{1,bör} = ?$
$Q_{2,max} = 30$	$Q_{2,min} = 5$	$Q_{2,bör} = ?$
$Q_{3,max} = 50$	$Q_{3,min} = 10$	$Q_{3,bör} = ?$
$Q_{4,max} = 70$	$Q_{4,min} = 15$	$Q_{4,bör} = ?$
$Q_{5,max} = 10$	$Q_{5,min} = 0.5$	$Q_{5,bör} = ?$

$$Q_{tot,bör} = Q_{n,bör} + Q_{n+1,bör}\, Q_{n+2,bör}\ldots..$$

$Q_{marginal}$ sätts till 0.5 * $Q_{\alpha,max}$ Operatören anger sitt önskade värde för systemet, $Q_{tot,bör}$ = 200 l/min. Funktionen börjar med att dividera arbetslasten jämn emellan sig med WD funktionen.

$$Q_{bör} = \frac{200}{5} = 20 \text{ l/min}$$

Här ligger flödes börvärdet antingen över sin maxkapacitet eller under marginalen för den. Vilka av villkor ska prioriteras? Ska man kontrollera om flödet är under marginalen eller om maximalt flödes kapacitet har överskridits? Vi sätter större tyngdpunkten på första frågan, om att kontrollera om börvärdet understiger marginal värdet, motiveringen finns i senare del av kapitel. Då värdet på börvärdet, 50 l/min, understiger marginalen. Systemet detekterar detta och börjar åtgärda det.

Viktigt att notera, vattenskäraren kan kanske för en okänd anledning, efterfråga efter ett flöde som är över Q_{min} men samtidigt också under $Q_{marginal}$. Problemet kan lösas genom att man låter jämförelsen $Q_{\alpha,\ bör}$ mindre än $Q_{marginal}$ och samtidigt $Q_{\alpha,\ bör}$ större än eller lika med Q_{min}. Denna kontroll görs endast i början av hela kalkyleringen, om den är sant, då ska drivenheten med den minsta maximala kapacitet arbeta om den har möjligheten till det, om alla har samma maximala kapacitet, tar man slaven med minst IP adress. Det är viktigt att notera, att denna funktion kanske inte behövs. Oftast eftersträvar man ett högt tryck.

Utför samma kontroller som i exempel ett, kontrollerar om drivenheterna kan hantera detta, annars lägger den till ytterligare drivenheter. När man har nått det önskade värdet sker nästa steg. Först kontrollerar man att $Q_{\alpha,\ bör}$ är större eller lika $Q_{marginal}$ och att $Q_{\alpha,\ bör}$ är större eller lika med Q_{min}. Om villkoret är falskt går vi vidare till nästa steg. I detta fall är det falskt. Skulle det hända att villkoret är sant, ska man ta slaven med den minsta maximala kapacitet och IP adress, som nämndes ovan, sätta sitt flöde till $Q_{bör}$ och ignorera $Q_{marginal}$.

$$Q_{\alpha,\ bör} = \frac{200}{5} = 20 \text{ l/min}$$

$Q_{4,max} >= Q_{bör}$?
FALSE
$Q_{2,max} + Q_{4,max} >= Q_{\alpha,\ bör}$?
FALSE
$Q_{2,max} + Q_{3,max} + Q_{4,max} >= Q_{bör}$?
FALSE
$Q_{2,max} + Q_{3,max} + Q_{4,max} + Q_{5,max} >= Q_{bör}$?
FALSE
$Q_{2,max} + Q_{3,max} + Q_{4,max} + Q_{5,max} >= Q_{bör}$?
FALSE
$Q_{1,max} + Q_{2,max} + Q_{3,max} + Q_{4,max} + Q_{5,max} >= Q_{bör}$?
TRUE

Man kontrollerar därefter vilka drivenheter man kan nollställa, samma kalkylering som i exempel ett. Värdet där villkoret blir falsk, återgår man till föregående värde där det villkoret fortfarande är sant.

$Q_{1,max} + Q_{2,max} + Q_{3,max} + Q_{4,max} + Q_{5,max} >= Q_{bör}$?
TRUE
$Q_{1,max} + Q_{2,max} + Q_{3,max} + Q_{4,max} >= Q_{bör}$?
TRUE
$Q_{1,max} + Q_{3,max} + Q_{4,max} >= Q_{bör}$?
TRUE
$Q_{1,max} + Q_{4,max} >= Q_{bör}$?
FALSE
$Q_{1,max} + Q_{3,max} + Q_{4,max} >= Q_{bör}$

Efter detta ska Mastern börja kalkylera fram börvärdet för varje drivenhet. WD funktionen beräknar detta.

$Q_{\alpha,\ bör} = 200/3 = 66.67$ l/min

Men nu kommer vi till problemet att börvärdet är alltför stor för vissa drivenheter. Då ska detta värdet skickas till funktionen Over Flow Control (OFC).

200-50 = 150 >=0
150-70 = 80 >=0
80 – 100 = -20 >=0
DONE

$Q_{bör} - Q_{\alpha,max} = Q_{ny,\ bör}$
$Q_{ny,\ bör} >= 0$

Obs! Negativ tjugo, betyder att vi inte använder full kapacitet. När kalkyleringen är klar, skickar mastern ut order till slavarna och verkställer det. Systemets parametrar kommer se ut så här:

$Q_{1,max} = 100$ $Q_{1,min} = 30$ $Q_{1,bör} = 80$ l/min
$Q_{2,max} = 30$ $Q_{2,min} = 5$ $Q_{2,bör} = 0$ l/min
$Q_{3,max} = 50$ $Q_{3,min} = 10$ $Q_{3,bör} = 50$ l/min
$Q_{4,max} = 70$ $Q_{4,min} = 15$ $Q_{4,bör} = 70$ l/min
$Q_{5,max} = 10$ $Q_{5,min} = 0.5$ $Q_{5,bör} = 0$ l/min

$Q_{tot,bör} = 200$ l/min

Egentligen skulle Q_1,Q_2 och Q_4 lika väl kunnat stå för arbetet. Då skulle alla drivenheter få arbeta i 100 %, vilket är att föredra, med det hade krävts att man använde ett kombinatoriskt system. Detta är en energioptimerings problem då vi inte kommer upp i 100 % effektivt arbete, fast det finns möjligheter till det.

Detta problem löses inte i det selektiva systemet. Vi får undersöka ett annat system.

4.10.3 Kombinatorisk Test

I det selektiva systemet, fick vi problemet att det kan hända att vi inte alltid kan köra 100 % i alla drivenheter, då våra villkor förhindrar detta. För att lösa detta problem, bör man i så fall testa olika kombinationer av drivenheter, därför undersöker vi om att införa ett kombinatoriskt system.

Det som menas med kombinatoriskt system, är att vi försöker hitta alla möjliga kombinationer som kan lösa att vi kan få ett $Q_{\alpha,\,bör}$ som får drivenheterna att arbeta i 100 %. Men samtidigt dyker fler problem upp, hur länge ska man söka efter kombinationer? Vilka villkor ska det finnas? Om det finns flera kombinationer som löser samma problem, hur ska vi prioritera? Vi börjar med lite grundläggande digital system, detta kan verka lite off-topic, men detta är baktanken till att lösa problematiken.

I den digitala världen finns det ettor och nollor, med dessa kan vi få olika kombinationer beroende på hur många bitar man har. T.ex. 4 bitar ger, 2^4 kombinationer. Genom denna princip skapar vi ett kombinatoriskt system.

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1

Figur 4.26. Kombinatorisk tabell, 4-bitar.

Den börjar med att prova alla möjliga kombinationer med drivenheterna, i detta fall finns det 2^5 kombinationer. Mastern måste kontrollera om drivenhetens/ drivenheternas totala Q_{max} överstiger $Q_{bör}$ eller är lika med $Q_{bör}$. Om det händer att $Q_{\alpha, max}$ överstiger $Q_{bör}$, spara man detta i ett tillfällig register. Om det istället sker att $Q_{\alpha, max}$ är lika med det önskade $Q_{bör}$ för systemet, avslutar man testet där och Mastern börjar dividera arbetet med WD och sedan skickar börvärdet svaret vidare OFC.

$Q_{5,max}$ $Q_{4,max}$ $Q_{3,max}$ $Q_{2,max}$ $Q_{1,max}$
0 + 0 + 0 + 0 + 0 > $Q_{bör}$ OR = = $Q_{bör}$
0 + 0 + 0 + 0 + 1 > $Q_{bör}$ OR = = $Q_{bör}$
0 + 0 + 0 + 1 + 0 > $Q_{bör}$ OR = = $Q_{bör}$
Osv.

Det största problemet med detta är att det är extremt tidsödande för ett tidkritiskt system. Om användaren har ett lågt antal drivenheter, kan det kombinatoriska testet löna sig, definitionen av lågt antal drivenheter är cirka fyra till fem. Men om användaren närmar sig ett större antal drivenheter, kommer denna process att bli extremt tidsödande. Om användaren har 14 motorer, betyder det finns 2^{14} kombinationer, dvs. 16384 kombinationer, vilket göra att användaren måste vara begränsad till ett antal drivenheter, vilket inte är målet. Eftersom ett kombinatorisk test är alltför tidsödande, förkastar vi denna typ av problemlösning.

4.10.4 Selektivitet med Kombinatorisk Test

Båda typen av problemlösning, medbringar problem. I selektiv test har vi problemet med att vi inte får den bästa energioptimeringen och i det kombinatoriska testet får vi ett extremt tidsödande process. Ett av delmålen i detta examensarbete är just energioptimering, som selektiva testet hade problemet med, för att lösa detta använder vi oss av en kombination av det selektiva och det kombinatoriska testet. Vi använder oss föregående exempel två i det selektiva testet med lite modifiering i slutet.

Ex 2:

$Q_{1,max} = 100$ $Q_{1,min} = 30$ $Q_{1,bör} = ?$
$Q_{2,max} = 30$ $Q_{2,min} = 5$ $Q_{2,bör} = ?$
$Q_{3,max} = 50$ $Q_{3,min} = 10$ $Q_{3,bör} = ?$
$Q_{4,max} = 70$ $Q_{4,min} = 15$ $Q_{4,bör} = ?$
$Q_{5,max} = 10$ $Q_{5,min} = 0.5$ $Q_{5,bör} = ?$

$Q_{tot,bör} = Q_{n,bör} + Q_{n+1,bör}$

$Q_{marginal}$ sätts till 0.5 * $Q_{\alpha,max}$ Operatören anger sitt önskade värde för systemet, $Q_{tot,bör}$ = 200. Funktionen börjar med att dividera arbetslasten jämn emellan sig med WD funktionen.

$Q_{bör} = \frac{200}{5}$ = 20 l/min

Här ligger flödes börvärdet antingen över sitt maxkapacitet eller under marginalen för den. Vilka av villkorna ska prioriteras? Ska man kontrollera om flödet är under marginalen eller om maximala flödes kapacitet har överskridits? Vi sätter större första kontrollpunkten på att kontrollera om börvärdet understiger marginal värdet, motiveringen finns i tidigare i texten. Då värdet på börvärdet, 50 l/min, understiger marginalen. Systemet märker detta och börjar åtgärda det.

Utför samma kontroller som i exempel ett, kontrollerar om drivenheterna kan hantera detta, annars lägger den till ytterligare drivenheter. När man har nått det önskade värdet sker nästa steg. Notera att Mastern läggs till sist, då vi vill undvika att systemets flöde blir beroende enbart av Mastern.

Kontroll om Q_{tot} ligger i marginalen måste först kontrolleras, får man ett sant svar, betyder det att vi nollställer och ger drivenheterna nya börvärden. Ett falskt svar ger oss möjligheten att fortsätta vidare i funktionen.

$Q_{bör}$ >= $Q_{marginal}$ AND $Q_{bör}$ >= Q_{min}?
FALSE

$Q_{4,max}$ >= $Q_{bör}$?
FALSE
$Q_{2,max}$ + $Q_{4,max}$ >= $Q_{bör}$?
FALSE
$Q_{2,max}$ + $Q_{3,max}$ + $Q_{4,max}$ >= $Q_{bör}$?
FALSE
$Q_{2,max}$ + $Q_{3,max}$ + $Q_{4,max}$ + $Q_{5,max}$ >= $Q_{bör}$?
FALSE
$Q_{2,max}$ + $Q_{3,max}$ + $Q_{4,max}$ + $Q_{5,max}$ >= $Q_{bör}$?
FALSE
$Q_{1,max}$ + $Q_{2,max}$ + $Q_{3,max}$ + $Q_{4,max}$ + $Q_{5,max}$ >= $Q_{bör}$?
TRUE

Man kontrollerar därefter vilka drivenheter man kan nollställa, samma kalkylering som i exempel ett. Värdet där villkoret blir falsk, återgår man till föregående värde där det villkoret fortfarande är sant.

$Q_{1,max} + Q_{2,max} + Q_{3,max} + Q_{4,max} + Q_{5,max} >= Q_{bör}$?
TRUE
$Q_{1,max} + Q_{2,max} + Q_{3,max} + Q_{4,max} >= Q_{bör}$?
TRUE
$Q_{1,max} + Q_{3,max} + Q_{4,max} >= Q_{bör}$?
TRUE
$Q_{1,max} + Q_{4,max} >= Q_{bör}$?
FALSE
$Q_{1,max} + Q_{3,max} + Q_{4,max} >= Q_{bör}$

När villkoret når ett falsk påstående, får vi information om att $Q_{1,max}$ och $Q_{4,max}$ är kritiska drivenheter för systemet, och att Q_5 är den sammanlänkade drivenheten som får systemet att gå ihop. I detta skedde använder vi oss av kombinatorisk test. Vi byter ut $Q_{3,max}$ mot alla drivenheter som är mindre än $Q_{3,max}$ i registret och som inte är kritisk för systemet.

$Q_{1,max} + Q_{5,max} + Q_{4,max} >= Q_{bör}$ && $Q_{5,max} < Q_{3,max}$?
FALSE
$Q_{1,max} + Q_{2,max} + Q_{4,max} >= Q_{bör}$ && $Q_{2,max} < Q_{3,max}$?
TRUE
END OF REGISTER

$Q_{1,max}$ = 100 Kritisk
$Q_{2,max}$ = 30
$Q_{3,max}$ = 50 Sammanlänkad
$Q_{4,max}$ = 70 Kritisk
$Q_{5,max}$ = 10

$Q_{A,max} + Q_{B,max} + Q_{C,max} >= Q_{bör}$ && $Q_{5,max} < Q_{3,max}$?
FALSE
$Q_{1,max} + Q_{2,max} + Q_{4,max} >= Q_{bör}$ && $Q_{2,max} < Q_{3,max}$?
TRUE
END OF REGISTER

De värdena som stämmer överens med villkoret, sparas i ett temporärt register. Man jämför värdena med varandra och tar ut den sammanlänkade drivenheten som ger ett resultat som är närmast det önskade $Q_{bör}$. Man byter därefter ut det gamla sammanlänkade drivenheten med den nya sammanlänkade drivenheten, i vårt fall blir det ett byte får $Q_{3,max}$ till $Q_{2,max}$.

$Q_{\alpha,\ bör} = 200/3 = 66.67$

Svaret skickas till OFC, vi får samma problem som i exempel två, vårt framtagna börvärde överskriver Q_{max} för vissa drivenheter.

200-30 = 170
170-70 = 100
100 – 100 = 0
DONE

Som vi kan se här, når vi ett resultat på 100 % för varje drivenhet, som är det önskvärda och mest optimala för det. Dock ärver det Selektiva med kombinatorisk test ett problem från just det kombinatorisk test. Att ju fler drivenheter vi har desto fler kombinationer kan man ha, men vi har begränsat antalet kombinationer, om vi använder samma exempel som tidigare med fem drivenheter, betyder det inte att vi har längre 2^5 kombinationer, utan 5 kombinationer, vissa kan även uteslutits. Då det selektiva testet låter oss utesluta vissa och säga vilka av drivenheterna måste vara med för att systemet ska kunna fungera samt vilken drivenhet som kan bytas ut.

Men löser vi frågan om energi optimering? Tänk om vi får ett fall där vi egentligen haft möjligheten att låta systemets arbetsbörda tas hand om Mastern. Enligt kalkyleringen som vi har satt upp kommer detta att ske men inte alla fall. Vi tar ett exempel.

Ex:

Användaren önskar sig ett flöde på 100 l/min, till förfoga har man fem drivenheter, Mastern i systemet har en maxkapacitet på 100 l/min medan resterande har 25 l/min. Om systemet använder sig av den Selektiva med kombinatorisk testet, kommer man få ett system som ser ut så här.

Selektivitet med kombinatorisk test → Q_1 = 0 l/min (0%)
Q_2 = 25 l/min (100 %)
Q_3 = 25 l/min (100 %)
Q_4 = 25 l/min (100 %)
Q_5 = 25 l/min (100 %)

Energieffe-ktivast → Q_1 = 100 l/min (100%)
Q_2 = 0 l/min (0 %)
Q_3 = 0 l/min (0 %)
Q_4 = 0 l/min (0 %)
Q_5 = 0 l/min (0 %)

Rent energimässigt, är detta inget att efterfodra, eftersom vi får större chans till flera förluster ju fler motorer vi har som kör. I detta fall hade Mastern varit perfekt tillämpning till att hantera arbetslasten, men vår funktion hindrar oss, då vi har konstruerat den på ett sådan vis att vi undviker att Mastern står för ensamt för arbetslasten. Med anledningen att vi inte vet hur lång tid det tar för systemet att återhämta sig vid en förlust av en Master.
Om vi ska ha möjligheten till att låta Mastern stå för den totala arbetsbördan i systemet, måste vi åta åtgärder. Själva tiden för system återhämtning finns det ingen tillgång till, då detta kan man endast få genom programmera ihop ett system. Problematiken i denna fråga är, hur ska vi åtgärda felet om Mastern försvinner och är den som står för hela arbetsbördan i systemet?

I kapitlet Worst Case Scenario, nämnde vi att man skulle kunna lösa problematiken genom att låta Mastern beräkna drivenheternas börvärde och ett worst case scenario värde. Skillnaden var att $Q_{\alpha,\ bör}$ är det värdet som Slaven ska ge i flöde, medan worst case scenario värdet, Q_{WCS}, sparas i Slaven och sätts som börvärde i Slaven endast om Mastern försvinner. Orsaken till detta är för att kompensera det förlorade flödet som uppstår när Mastern förloras. Vi nämnde att man skulle använda en funktion till att göra detta möjligt, men nämnde inte riktigt hur själva uppbyggnaden ser ut, detaljerna finns i fallet *Marginal*.

Vi tar ett sista exempel i detta fall för att komma tillrätta med problemet, samt utökar funktionen selektivitet med kombinatorik test. Viktigt att notera att WCS beräknas efter LFC och OFC har exekverat med värdena från de. Vi tar ett exempel där Mastern får ha en stor del av arbetet och kan även tänka sig att arbeta ensam.

Ex:

$Q_{1,max} = 100$ $Q_{1,min} = 30$ $Q_{1,bör} = ?$
$Q_{2,max} = 30$ $Q_{2,min} = 5$ $Q_{2,bör} = ?$
$Q_{3,max} = 50$ $Q_{3,min} = 10$ $Q_{3,bör} = ?$
$Q_{4,max} = 70$ $Q_{4,min} = 15$ $Q_{4,bör} = ?$
$Q_{5,max} = 10$ $Q_{5,min} = 0.5$ $Q_{5,bör} = ?$

$$Q_{tot,bör} = Q_{n,bör} + Q_{n+1,bör}$$

Användare efterfråga ett totalt flöde i systemet som är 150 l/min. Till förfoga har han fem drivenheter, med sin respektive maximala och minimala kapacitet. Mastern får i detta fall ordern då att få ett flöde på 150 l/min i systemet. I detta fall är $Q_{marginal}$ femtio procent av $Q_{\alpha,\ max}$.
Det första som sker är WD funktionen, som jämnfördelar arbetet mellan drivenheterna.

$Q_{\alpha,\ bör} = \frac{150}{5} = 30$ l/min

Mastern kontrollerar för varje drivenhet om detta börvärde understiger ett flödes marginal. Om detta sker, aktiveras funktionen för att återgärda felet (LFC). Först kontrollerar man dock om det totala flödet är under en drivenhets marginal men samtidigt också över ens minimala kapacitet, står med detaljerad i fallet *Marginal*.

$Q_{\alpha,\ bör} <= Q_{marginal}$ AND $Q_{\alpha,\ bör} >= Q_{min}$?
FALSE

Beta = flödesmarginal

$Q_{1,\ max} >= Q_{\alpha,\ bör} /\beta$
FALSE
$Q_{1,max} + Q_{4,max} >= Q_{\alpha,\ bör} /\beta$?
TRUE

När detta villkor stämmer, får vi information om vilka drivenheter vi kan nollställa, men innan vi gör detta, testar vi om vi kan byta ut en drivenhet mot en annan för att det ska bli mer energieffektivt, i detta fall blir det drivenhet Q_4. Vi byter dock endast ut den drivenhet som har lägst maximala kapacitet **om** det finns en sådan, annars hoppar vi över detta moment. Villkoret för detta blir:

$Q_{1,\ max} + Q_{5,\ max} >= Q_{\alpha,\ bör}$ (AND $Q_{5,max} < Q_{4,max}$)?
FALSE
$Q_{1,\ max} + Q_{2,\ max} >= Q_{\alpha,\ bör}$ (AND $Q_{5,max} < Q_{4,max}$)?
FALSE
$Q_{1,\ max} + Q_{3,\ max} >= Q_{\alpha,\ bör}$ (AND $Q_{3,max} < Q_{4,max}$)?
TRUE

$Q_{1,max}$ = 100 Störst (ska ej bytas ut)
$Q_{2,max}$ = 30
$Q_{3,max}$ = 50
$Q_{4,max}$ = 70 Bytbar
$Q_{5,max}$ = 10

Om vi kommer fram till villkor som är sanna, sparas detta i ett register och man tar ut det värdet som är minst. Uppstår det att det finns två eller flera kombinationer som har det minsta värdet, ska listan sorteras enligt minsta börvärdet och ta den kombinationen som finns på första plats.

När vi har provat de kombinationer som finns, om de finns, nollställer vi de drivenheter som inte finns med i kombinationen.

$Q_{1,max}$ = 100 liter/min
$Q_{2,max}$ = 0 l/min
$Q_{3,max}$ = 50 l/min
$Q_{4,max}$ = 0 l/min
$Q_{5,max}$ = 0 l/min

Detta exempel är väldigt snarligt de exemplen vi har använt oss av i *Selektivitet med Kombinatorisk Test*, men här är skillnaden att vi inte undviker att systemet blir beroende av Mastern. Vi föregående tillät vi inte detta, vilket bidrog till att vi gjorde onödiga kalkyleringar, där vi gick fram och tillbaks, vilket tar ytterligare tid.

Men för att använda detta måste vi ha möjligheten att kompensera vid en eventuell förlust av Mastern och sitt flöde. Detta gör vi genom att använda oss av WCS funktionen, som nämnts tidigare. Genom att använda WCS kan vi kompensera denna tänkbara förlust, funktionen står mer detaljerad i fall *WCS*.

Dock måste man ha i tanken att kunden kanske inte är ute efter att ha det mest energioptimalaste lösningen, utan efter det mest arbetsmiljövänliga lösning. Vi får inte glömma att en motor bidrar till oväsen, beroende på vilken styrka den arbetar med. I detta fall sätter vi två drivenheter till att arbeta 100 %, vilket gör att oväsen kommer vara stora, men man kan använda sig av funktionen ändå, motiveringen kommer längre fram.

I det fallet är WD funktionen lämpligast att använda och endast den, för på så sätt minskar vi ljusstörningar i arbetsmiljön då arbetslasten är jämnfördelad och alla maskiner bidrar till mindre oväsen. Maskinoperatören får själv välja vilket mål man vill uppnå, ska systemet vara energioptimal eller arbetsmiljövänlig? När systemet körs, måste maskinoperatören själv välja detta, enklaste för tillmötes gå detta, är att vi helt enkelt sätter en meny på displayen som låter användaren välja.

Vi har nämnt hur funktionen ska hantera om det finns drivenheter med olika kapaciteter, men inte nämnt hur funktionen är tänkt att lösa börvärde frågan om alla drivenheter har samma kapacitet. Problemet med exemplet vi gick

igenom nyss, är att funktionen bidrar med mycket oväsen. Då en motor bidrar med mer oväsen ju större arbetslast den har.

Kan vi fortfarande använda funktionen? Det kan vi, men vi måste ha ytterligare ett villkor. Nämligen, **om** det inte finns någon drivenhet med en mindre kapacitet, då kan vi säkerställa att vi kan fördela arbetet emellan sig genom arbetsfördelning. Vi tar ett exempel.

Ex:

$Q_{1,max} = 100$	$Q_{1,min} = 30$	$Q_{1,bör} = ?$
$Q_{2,max} = 100$	$Q_{2,min} = 30$	$Q_{2,bör} = ?$
$Q_{3,max} = 100$	$Q_{3,min} = 30$	$Q_{3,bör} = ?$
$Q_{4,max} = 100$	$Q_{4,min} = 30$	$Q_{4,bör} = ?$
$Q_{5,max} = 100$	$Q_{5,min} = 30$	$Q_{5,bör} = ?$

Alla drivenheter i detta exempel har en maximalkapacitet på 100 l/min. Systemet efterfrågar ett flöde i systemet som ligger på 120 l/min i en viss tidsperiod. Som systemet är designad nu, kommer köra funktionen selektivitet med kombinatorisk test, fast utan det kombinatoriska testet. Då det inte finns andra drivenheter med mindre kapacitet. Om inte det kombinatoriska testet ska exekveras, då kan vi säkerställa att vi kan fördela arbetet jämnt emellan drivenheterna, då alla har samma kapacitet. Vi testar det teoretiska programmet.

$Q_{1,\,max} >= Q_{bör}/\beta$
FALSE
$Q_{1,max} + Q_{2,max} >= Q_{bör}/\beta$?
TRUE

Programmet körs, därefter har den hittat de drivenheter som kan tänkas sättas i arbete. Det kombinatoriska testet aktiveras **inte** då kravet inte är uppfyllt, nämligen att vi har olika kapacitet i systemet. Detta detekteras och vi kan då använda funktionen WD igen, för att fördela arbetet.

$$Q_{\alpha,\,bör} = \frac{120}{2} = 60 \text{ l/min}$$

Drivenheter ett till två kommer att ha ett börvärde som är på 60 l/min, medan resterande nollställs.

$Q_{1,max} = 100$	$Q_{1,min} = 30$	$Q_{1,bör} = 60$ l/min
$Q_{2,max} = 100$	$Q_{2,min} = 30$	$Q_{2,bör} = 60$ l/min
$Q_{3,max} = 100$	$Q_{3,min} = 30$	$Q_{3,bör} = 0$ l/min
$Q_{4,max} = 100$	$Q_{4,min} = 30$	$Q_{4,bör} = 0$ l/min
$Q_{5,max} = 100$	$Q_{5,min} = 30$	$Q_{5,bör} = 0$ l/min

Som vi ser här, låter vi drivenheterna ett och två arbeta i 60 %. Genom att använda denna funktion har vi dessutom minskat oväsendet då vi inte kör på 100 %. Men då kommer frågan, är det inte bättre att enbart att exekvera funktionen WD och fördela arbetet jämnt emellan sig, för att minska på bullret? Jo, det är sant, men samtidigt får inte glömma att ett överdimensionerad motor som inte kör på sitt nominella varvtal, kan lika väl ge större förluster än en mindre motor som kör på kör på sitt nominella varvtal. Funktionen är till för att balansera detta, och våra teoretiska beräkningar visar detta.

Vi har inte nämnt hur detekteringen ska gå till och det gör vi nu. Det är tänkt att funktionen använder drivenheten med den minsta IP adress och därefter jämför man det maximala kapacitet emellan varje drivenhet. Om det finns en skillnad, då ska funktionen använda sig av ett selektivitet med kombinatoriska testet, om det inte finns en skillnad, exekverar vi samma funktion fast utan ett kombinatorisk test, istället använder vi oss av WD funktionen.

Som vi har visat här, når man målet för energioptimering. Men eftersom vi har dynamiskt flöde, är detta inte passande, eftersom man kommer vara tvungen att växla drivenheterna på och av, vilket också ger förluster. Om denna lösning skulle tänkas använda, måste man utveckla funktionen ytterligare, men eftersom en drivenhet har ett RAM-minne i megabyte skalan, måste undvika denna typ av lösning då det kan ta alltför lång tid att kalkylera ut ett nytt börvärde för alla drivenheter.

4.10.5 Easy Save

Vi har diskuterat med vår handledare om funktionen LFC, slutsatsen vi kom fram till var att man uppfyllde målet bättre i energioptimeringsdelen. Men var oerhört svårt att implementera och ett nytt problem uppstod, nämligen att om funktionen fungerar som det är tänkt, kommer många asynkronmotorer att stängas av och startas.

För att lösa detta problem, har vi diskuterat att vidare utveckla LFC metoden för att lösa denna sorts problematik. Dock är det fortfarande svårt att implementera, oavsett om vi löser problemet, eftersom varje drivenhet har en begränsning av hur mycket den kan kalkylera, en drivenhet har ett RAM-minne på megabyte skalan.
I analysering av fallet LFC, tog vi inte hänsyn till implementation av funktionen, utan koncentrerade oss mer på energioptimerings frågan, vilket gav upphov till ett mer energioptimerad system, men samtidigt väldig svår

implementation. I detta fall undersöker vi om det är möjligt att implementera ett enklare program, men samtidigt uppnå målet med energioptimering. Dock är det viktigt att notera, att vi använder oss av våra tidigare undersökningar, men med modifikationer.

Undersökningen av LFC funktionen, har gett oss viktig information om hur vi på bästa sätt kan energieffektivisera. Easy Save funktionen, som vi förklarar alldeles snart hur den är tänkt att fungerar, är en nerskalad version av funktionen LFC.
Istället för att använda oss av kombinatorik, använder vi oss endast av det selektiva testet men med olika prioriteringar. Vi nämnde tidigare i fallet LFC att vi tar drivenheten med största motorkapaciteten och adderar det med andra drivenheter som har en stor kapacitet jämfört med den första drivenheten, mer information finns i fallet LFC.

Problem som dyker upp, vilket villkor ska gälla för att nollställa en motor? Vilka motorer bör arbeta? Hur löser vi problemet med att systemet stänger av och startar motorn kontinuerligt?
Easy Save är tänkt att fungera på samma princip, men har helt andra villkor och använder sig inte av kombinatorik. Det första som sker är att man tar drivenheten med näst högsta kapacitet jämfört med börvärdet och sedan adderar drivenheten med minsta kapacitet i systemet. Orsaken till att vi inte använder drivenheten med den maximala kapacitet, är för att vi eftersträvar en flödesmarginal.

Principen här är att vi startar näst största motorn i systemet, därefter kontrollerar vi om den kan möta efterfrågan, om detta inte kan göras adderar vi drivenheten med minsta kapacitet i systemet också vidare.
För att förenkla förståelsen av funktionen, tar vi två exempel, notera dock att vi undersöker även fallet att vi tillåter drivenheten med maximal kapacitet att tilldelas först.

Ex 1:

$Q_{1,max} = 100$	$Q_{1,min} = x$	$Q_{1,bör} = ?$ l/min
$Q_{2,max} = 80$	$Q_{2,min} = x$	$Q_{2,bör} = ?$ l/min
$Q_{3,max} = 60$	$Q_{3,min} = x$	$Q_{3,bör} = ?$ l/min
$Q_{4,max} = 40$	$Q_{4,min} = x$	$Q_{4,bör} = ?$ l/min
$Q_{5,max} = 20$	$Q_{5,min} = x$	$Q_{5,bör} = ?$ l/min

Vid ett visst moment, eftersträvas ett visst tryck, som ges av ett flöde, $Q_{bör}$, 150 l/min. I systemet kontrolleras först vem av drivenheterna som har den näst största maximala kapaciteten, i detta fall är det drivenheten med flödet $Q_{2,\,max}$.

Därefter kontrolleras om drivenheten kan tillgodose behovet, om inte, adderas den drivenheten med minsta kapacitet i uträkningen. Om villkoret $Q_{max, tot}$ är större än eller lika med $Q_{bör}$ stämmer, då kan vi använda dessa till att sätta i arbete. Vi illustrerar detta för att underlätta förståelsen.

$Q_{bör}$ = 150 l/min (Eftersträvad flöde)	$Q_{2, max}$ >= $Q_{bör}$ (False) Lägger till den drivenhet med minsta kapacitet som inte arbetar $Q_{2, max} + Q_{5, max}$ >= $Q_{bör}$ (True) Fortsätter till nästa steg

Vi bör även kontrollera om vi kan nollställa en drivenhet. Denna kontroll görs först när villkoret ovan stämmer. Principen med kontrollen, är att vi undersöker om vi kan tillåta de större motorerna att arbeta och stänga av de mindre. Kontrollen sker nedan.

if ($Q_{bör, ny} > 0$) (True)
$Q_{bör, ny} = Q_{bör} – Q_1$ = 50 l /min

if ($Q_{bör, ny} > 0$) (True)
$Q_{bör, ny} = Q_{bör, ny} – Q_5$ = 30 l /min

if ($Q_{bör, ny} > 0$) (True)
$Q_{bör, ny} = Q_{bör, ny} – Q_4$ = -10 l /min

if ($Q_{bör, ny} > 0$) (False)
Avsluta kontrollen

Med hjälp av den här beräkningen kan vi få informationen om vilka motorer som kan nollställas. I detta fall blir det drivenheten med flödeskapacitet $Q_{2, max}$ och $Q_{3, max}$ som ska nollställas. Medan drivenheterna med flödena $Q_{1, max}$, $Q_{4, max}$ och $Q_{3, max}$ sätts i arbete.

$Q_{1,max}$ = 100	$Q_{1,min}$ = x	$Q_{1,bör}$ = 100 l/min
$Q_{2,max}$ = 80	$Q_{2,min}$ = x	$Q_{2,bör}$ = 0 l/min
$Q_{3,max}$ = 60	$Q_{3,min}$ = x	$Q_{3,bör}$ = 0 l/min
$Q_{4,max}$ = 40	$Q_{4,min}$ = x	$Q_{4,bör}$ = 40 l/min
$Q_{5,max}$ = 20	$Q_{5,min}$ = x	$Q_{5,bör}$ = 20 l/min

Vid nästa beräkning används samma drivenheter igen som föregående beräkning för att kontrollera om de kan hantera det nya efterfrågade Q_{tot}, som är nu 170 l/min. Samma steg utförs som i föregående.

$Q_{bör}$ = 170 l/min (Eftersträvad flöde)	$Q_{1,\,max} + Q_{5,\,max} + Q_{4,max} >= Q_{bör}$ (False) Lägger till den drivenhet med minsta kapacitet som inte arbetar $Q_{1,\,max} + Q_{5,\,max} + Q_{4,max} + Q_{3,\,max} >= Q_{bör}$ (True) Fortsätter till nästa steg

Kontroll:

if ($Q_{bör,\,ny} > 0$) (True)
$Q_{bör,\,ny} = Q_{bör} - Q_1$ = 70 l /min

if ($Q_{bör,\,ny} > 0$) (True)
$Q_{bör,\,ny} = Q_{bör,\,ny} - Q_3$ =10 l /min

if ($Q_{bör,\,ny} > 0$) (True)
$Q_{bör,\,ny} = Q_{bör,\,ny} - Q_4$ = - 30 /min

if ($Q_{bör,\,ny} > 0$) (False)
Avsluta kontrollen

$Q_{1,max}$ = 100	$Q_{1,min}$ = x	$Q_{1,bör}$ = 100 l/min
$Q_{2,max}$ = 80	$Q_{2,min}$ = x	$Q_{2,bör}$ = 0 l/min
$Q_{3,max}$ = 60	$Q_{3,min}$ = x	$Q_{3,bör}$ = 60 l/min
$Q_{4,max}$ = 40	$Q_{4,min}$ = x	$Q_{4,bör}$ = 10 l/min
$Q_{5,max}$ = 20	$Q_{5,min}$ = x	$Q_{5,bör}$ = 0 l/min

Vid en senare tidpunkt krävs det ett flöde på 290 l/min, vilket kräver att alla drivenheter sätts i arbete.

$Q_{bör}$ = 290 l/min (Eftersträvad flöde)	$Q_{1,\,max} + Q_{4,\,max} + Q_{3,max} >= Q_{bör}$ (False) Lägger till den drivenhet med minsta kapacitet som inte arbetar $Q_{1,\,max} + Q_{4,\,max} + Q_{3,max} + Q_{5,\,max} + Q_{2,max} >= Q_{bör}$ (False) Lägger till den drivenhet med minsta kapacitet som inte arbetar $Q_{2,\,max} + Q_{4,\,max} + Q_{3,max} + Q_{5,\,max} + Q_{1,max} >= Q_{bör}$ (True) Fortsätter till nästa steg

Som exemplet visar, behåller vi samma drivenheter som innan $Q_{bör}$ ändrades. Men vad händer om vi får ett plötsligt tryckfall, vilket gör att vi endast behöver ett flöde på 20 l/min? Samma steg utförs som i föregående exempel.

$Q_{bör}$ = 20 l/min

(Eftersträvad flöde)

$Q_{2,\,max} + Q_{4,\,max} + Q_{3,max} + Q_{5,\,max} + Q_{1,max} >= Q_{bör}$ (True)

if ($Q_{bör,\,ny} >= 0$) (True)
$Q_{bör,\,ny} = Q_{tot} - Q_2$ = -60 l /min
Avsluta kontrollen

I detta fall, nollställs drivenheterna med flödet $Q_{4,\,max}$, $Q_{3,max,}$ $Q_{5,\,max}$, $Q_{1,max}$. Medan drivenheten med flödet $Q_{2,\,max}$, får arbeta ensamt.
Problematiken här, är att vi inte får det mest optimala lösningen. Istället för att låta drivenheten med $Q_{2,\,max}$ arbeta 20 procent (20 % av 80 = 16), hade vi egentligen velat använda drivenheten med flödet $Q_{5,max}$, att arbeta. Då denna kommer till sin nominella arbetslast (100 %). Om vi hade tillåtit detta att ske, får vi samma problematik som i LFC funktionen, nämligen att drivenheter kommer stänga av och starta sina motorer, helt enkelt kommer de att switchas frekvent. För att undvika denna sorts av problematik, tillåter vi inte att detta sker.

Samtidigt måste programmet kunna kontrollera vilken drivenhet som är näst störst i systemet. Om det finns drivenheter med lika stor maximal flödeskapacitet, måste man använda sig av IP prioritering, nämligen att man tar drivenheten med den minsta IP adressen, som vi har stött på tidigare.

Vi undersöker även om vi istället tillåter drivenheten med den största flödeskapaciteten att kontrolleras först, och jämför detta med drivenheten med näst störst kapacitet

Ex:

$Q_{1,max}$ = 100 $Q_{1,min}$ = 30 $Q_{1,bör}$ = ? l/min
$Q_{2,max}$ = 80 $Q_{2,min}$ = 20 $Q_{2,bör}$ = ? l/min
$Q_{3,max}$ = 60 $Q_{3,min}$ = 10 $Q_{3,bör}$ = ? l/min
$Q_{4,max}$ = 40 $Q_{4,min}$ = 5 $Q_{4,bör}$ = ? l/min
$Q_{5,max}$ = 20 $Q_{5,min}$ = 1 $Q_{5,bör}$ = ? l/min

Vattenskäraren efterfrågar under en viss tidperiod ett visst tryck. Detta tryck motsvarar ett flöde på motsvarande x l/min, och förändras i varje tidsperiod. I

beräkningen finns det två metoder och används för att jämföra dessa. (1) Max kapacitet, (2) Näst störst max kapacitet.

$Q_{bör}$ = 80 l/min	(1) $Q_{1, max} >= Q_{bör}$ (True)
$Q_{bör}$ = 100 l/min	(1) $Q_{1, max} >= Q_{bör}$ (True)
$Q_{bör}$ = 120 l/min	(1) $Q_{1, max} + Q_{5, max} >= Q_{bör}$ (True)
$Q_{bör}$ = 150 l/min	(1) $Q_{1, max} + Q_{5, max} + Q_{4, max} >= Q_{bör}$ (True)
$Q_{bör}$ = 250 l/min	(1) $Q_{1, max} + Q_{5, max} + Q_{4, max} + Q_{3, max} + Q_{2, max} >= Q_{bör}$ (True)

$Q_{bör}$ = 80 l/min	(2) $Q_{2, max} >= Q_{bör}$ (True)
$Q_{bör}$ = 100 l/min	(2) $Q_{2, max} + Q_{5, max} >= Q_{bör}$ (True)
$Q_{bör}$ = 120 l/min	(2) $Q_{2, max} + Q_{5, max} + Q_{4, max} >= Q_{bör}$ (True)
$Q_{bör}$ = 150 l/min	(2) $Q_{2, max} + Q_{5, max} + Q_{4, max} + Q_{3, max} >= Q_{bör}$ (True)
$Q_{bör}$ = 250 l/min	(2) $Q_{2, max} + Q_{5, max} + Q_{4, max} + Q_{3, max} + Q_{1, max} >= Q_{bör}$ (True)

Jämförelsen visar att om vi använder principen ”näst störst jämfört med börvärdet”, måste vi ta till fler drivenheter än vad vi egentligen behöver. Dessutom måste vi göra ytterligare kontroller, om vem som näst störst (behövs också i principen ”störst kapacitet”) och när vi nollställer drivenheter, måste vi kontrollera att vi inte nollställer den näst största drivenheten jämfört med

börvärdet. Men vi får samma problematik som innan, nämligen att systemet kommer att starta och stänga av motorer kontinuerligt.

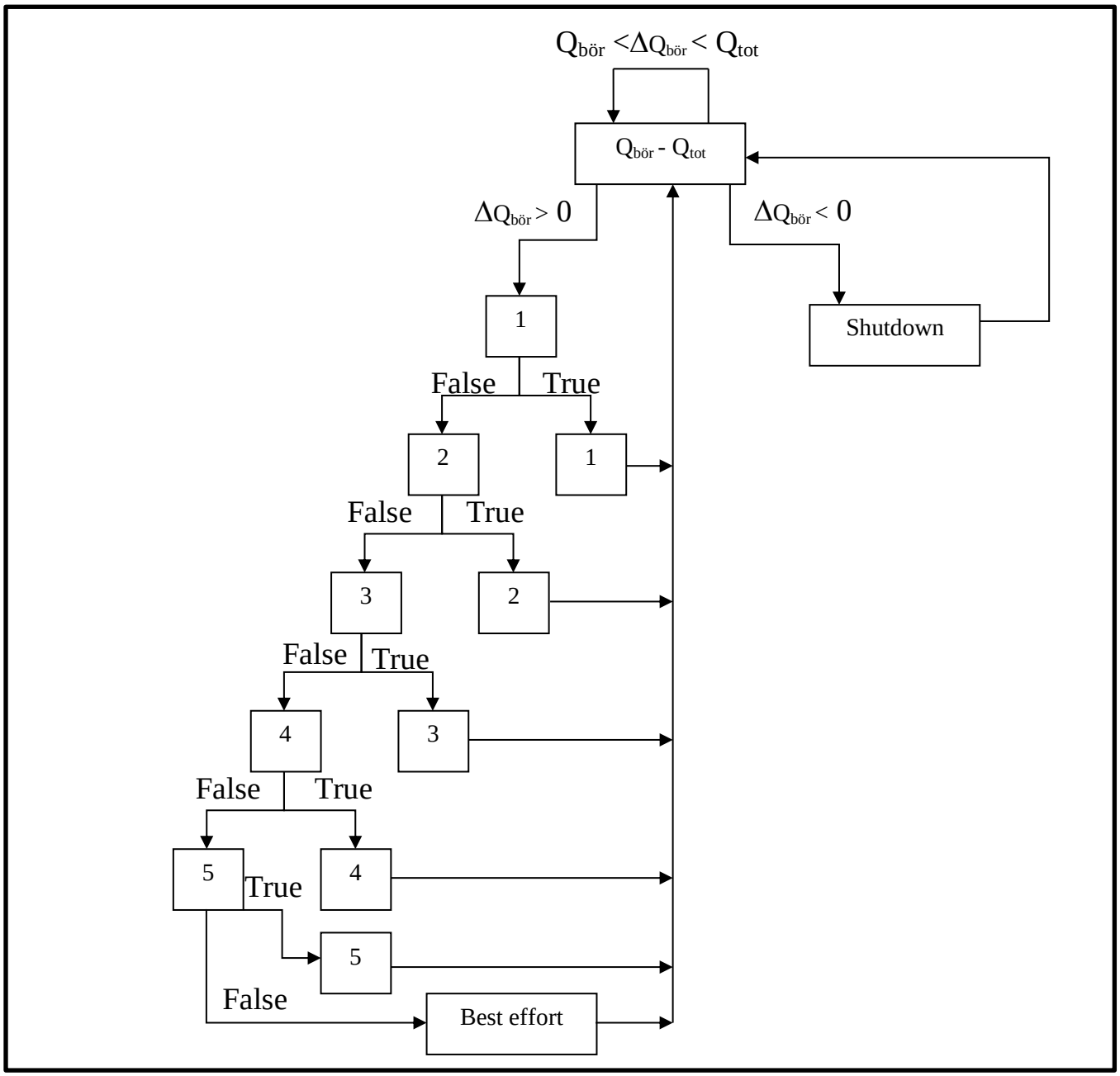
Vi har fram tills nu koncentrat oss på att systemet ska se till att det önskade flödet ska verkställas med motorer som är förhållandevis nära eller på sin nominella arbetslast, för att undvika överdimensionering.
Problemet med detta är att vi kommer starta och stänga av motorer kontinuerligt, man får inte glömma att det tar några sekunder innan motorn når det önskade varvtalet. I ett system som är tidskritisk kan man inte ha ett sådant system, eftersom om det efterfrågade flödet inte verkställs snabbt av systemet, måste man kassera produkten.

Fram tills nu, har våra undersökningar visat och bevisat att ha prioriteten att undvika överdimensionering inte är möjligt, när vi har ett tidskritisksystem. Men vi måste fortfarande ha detta i tanken när vi utvecklar vidare algoritmen.

4.10.6 Flow scheme version 1.0

Efter ett möte med vår handledare visade sig att easy save lösningens nackdelar var stora och implementationen kunde göras lättare, bestämde vi oss för att föra vidare utvecklingen på en bättre lösning. Vi fick bland annat tips under mötet att använda oss utav så kallad flödesschema, som beskriver stegvist med olika alternativa vägar en process tar för att underhålla systemet i olika situationer. Innan vi började med att diskutera och undersöka hur vårt flödesschema skulle se ut, fick vi hjälp och tips av vår handledares kollega, hur en möjlig struktur på schemat kan se ut.
När vi väl satte oss in i utvecklingen av schemat fick vi även tips på ett sätt man kan söka efter rätt motor att sköta om en viss kapacitet och med så få motorer som möjligt, samtidigt som vi vill ha ett system där överdimensionering inte inträffar. Något annat viktigt att påpeka är som vi nämnt tidigare att vi vill starta och stänga av motorer så sällan som möjligt också. Lösningen förslaget för algoritmen flow scheme v 1.0:

Figur 4.27. Arkitektur för algoritmen.



I första steget längst upp, tar vi skillnaden av subtraktionen mellan $Q_{bör}$ och Q_{tot}, kontrollerar vi tryckändringar. beroende på om vårt flödes börvärde är större eller mindre än systemets totala flöde, kan vi antigen öka antalet jobbande motorer eller möjligtvis stänga av.

När systemet startas för första gången, kan man detektera att man inte kan tillfredsställa systemets efterfråga, då man inte har några arbetande pumpar. Det första som sker då är att man kontrollerar om en motor klarar av

efterfrågan, annars väljer man den största motorn och går vidare, kontrollerar en andra motor, låter dessa två arbeta tillsammans om dem uppfyller efterfrågan. Skulle dessa två ej uppfylla vårt önskade flöde, kommer man ta den näst största motorn och gå över till letandet av en tredje motor som kan tillsammans med dem andra två kan komma över börvärdet.

Om ett tryckfall uppstår, dvs. att vi nu fått ett lägre börvärde än det tidigare, efter att systemet lyckats komma upp till det gamla börvärdet, kommer minsta motorn att kontrolleras först om denna kan stängas av utan att vi understiger det nya börvärdet. Sedan kontrolleras näst minsta och denna princip körs tills kontrollen visar att totala flödet är under börvärdet. Skulle vi t.ex. ha två körandes motorer när ett trycknedfall uppstår, kontrollerar vi motor två som är mindre än motor ett, om denna kan stängas av utan att vi hamnar under börvärdet, skulle detta ej vara möjligt då får vi istället fortfarande köra på två motorer men istället försöka hitta en mindre motor som kanske kan ersätta motor två.

För att underlätta förståelsen tar vi också ett praktiskt exempel:
Låt oss säga att vi har ett $Q_{bör}$ på 220 liter per minut och att vi har fem motorer som har maximala flöden på 100, 80, 60, 40, 20. Notera att $\Delta Q_{bör}$ är vårt nya bördvärde medan $Q_{bör}$ är det föregående.

$Q_{1,max} = 100$			
$Q_{2,max} = 80$			
$Q_{3,max} = 60$			
$Q_{4,max} = 40$			
$Q_{5,max} = 20$			
Antal motorer i drift	1	2	3

Vi börjar med att kontrollera om en motor klarar av att ta ett flöde på 220 liter per minut, systemet kommer att markera ut en ”0” där systemet testat och markerar ut ”1” om motor klarar av laster eller om motor är den sista och största i dennas kolumn, även om denna sista motor inte klarar av lasten. Systemet kontrollerar alltid den minsta först och jobbar sig upp till den högsta:

$Q_{1,max} = 100$	1		
$Q_{2,max} = 80$	0		
$Q_{3,max} = 60$	0		
$Q_{4,max} = 40$	0		
$Q_{5,max} = 20$	0		
Antal motorer	1	2	3

Eftersom en motor inte kan hantera det önskade värdet ensam, inser nu systemet att vi behöver lägga och kontrollera ytterligare en motor, där den föregåendes motor och den vi nu ska kontrolleras flöde kommer att adderas ihop:

$Q_{1,max} = 100$	1	0	
$Q_{2,max} = 80$	0	1	
$Q_{3,max} = 60$	0	0	
$Q_{4,max} = 40$	0	0	
$Q_{5,max} = 20$	0	0	
Antal motorer	1	2	3

Notera att Q_1 markeras med ”0” i den andra kolumnen, då vi redan ställt den på ”1” i föregående kolumn. Därför markerar vi Q_2 med hänsyn till vår princip som vi nämnde tidigare.

Efter att vi nu tagit både Q_1 och Q_2, adderat ihop dem, får vi ett flöde på 180 liter per minut, detta ligger fortfarande under vårt börvärde och därför måste vi ännu en gång testa ytterligare en motor till.

$Q_{1,max} = 100$	1	0	
$Q_{2,max} = 80$	0	1	
$Q_{3,max} = 60$	0	0	
$Q_{4,max} = 40$	0	0	1
$Q_{5,max} = 20$	0	0	0
Antal motorer	1	2	3

I den tredje kolumnen kontrollerade vi våra tidigare motorer med Q_4 som uppfyllde kravet större eller lika med flödes börvärdet 220 liten per minut, nu när vår kontroll hittat dem lämpliga motorerna startas dessa tre och arbetar tills börvärdet ändras, då sker en ny likadan kontroll igen.

Skulle nu börvärdet sjunka, låt oss säga till 60 liter per minut, kommer kontroller att göras baklänges istället:

$Q_{1,max} = 100$	1	0	
$Q_{2,max} = 80$	0	1	
$Q_{3,max} = 60$	0	0	
$Q_{4,max} = 40$	0	0	0
$Q_{5,max} = 20$	0	0	0
Antal motorer	1	2	3

Första kontrollen görs på den minsta motorn som i detta fall är motor in kolumn tre, kollar om det finns möjlighet att stänga den tredje motorn(Q_4) utan att man understiger börvärdet, i detta fall sjunker vi till 180 liter per minut, vilket ger oss ett ”TRUE” att stänga av tredje motorn.

$Q_{1,max}$ = 100	1	0	
$Q_{2,max}$ = 80	0	0	
$Q_{3,max}$ = 60	0	0	
$Q_{4,max}$ = 40	0	0	0
$Q_{5,max}$ = 20	0	0	0
Antal motorer	1	2	3

Sedan kontrolleras nästa minsta som är den andra motorn och då sjunker vi till 100 liter per minut, vilket fortfarande är större än vårt börvärde 60 liter per minut, därför får vi även här ett ”TRUE” att stänga av den. Eftersom vi nu endast har en körande motor, försöker vi hitta en motor som är mindre än Q_1 och som är större eller lika med börvärdet. Vi testar Q_2 och hamnar på 80, kravet uppfylls så vi går ett steg ner igen och testar 60, som fortfarande ger oss ”TRUE” till vi får ett ”FALSE” på Q_4. När vi hittat vår gräns via vårt ”FALSE”, återgår vi då till vårt senaste ”TRUE” som är Q_3 i detta fall och startar denna motor, medan Q_1 som ger oss 100 i flöde stängs.

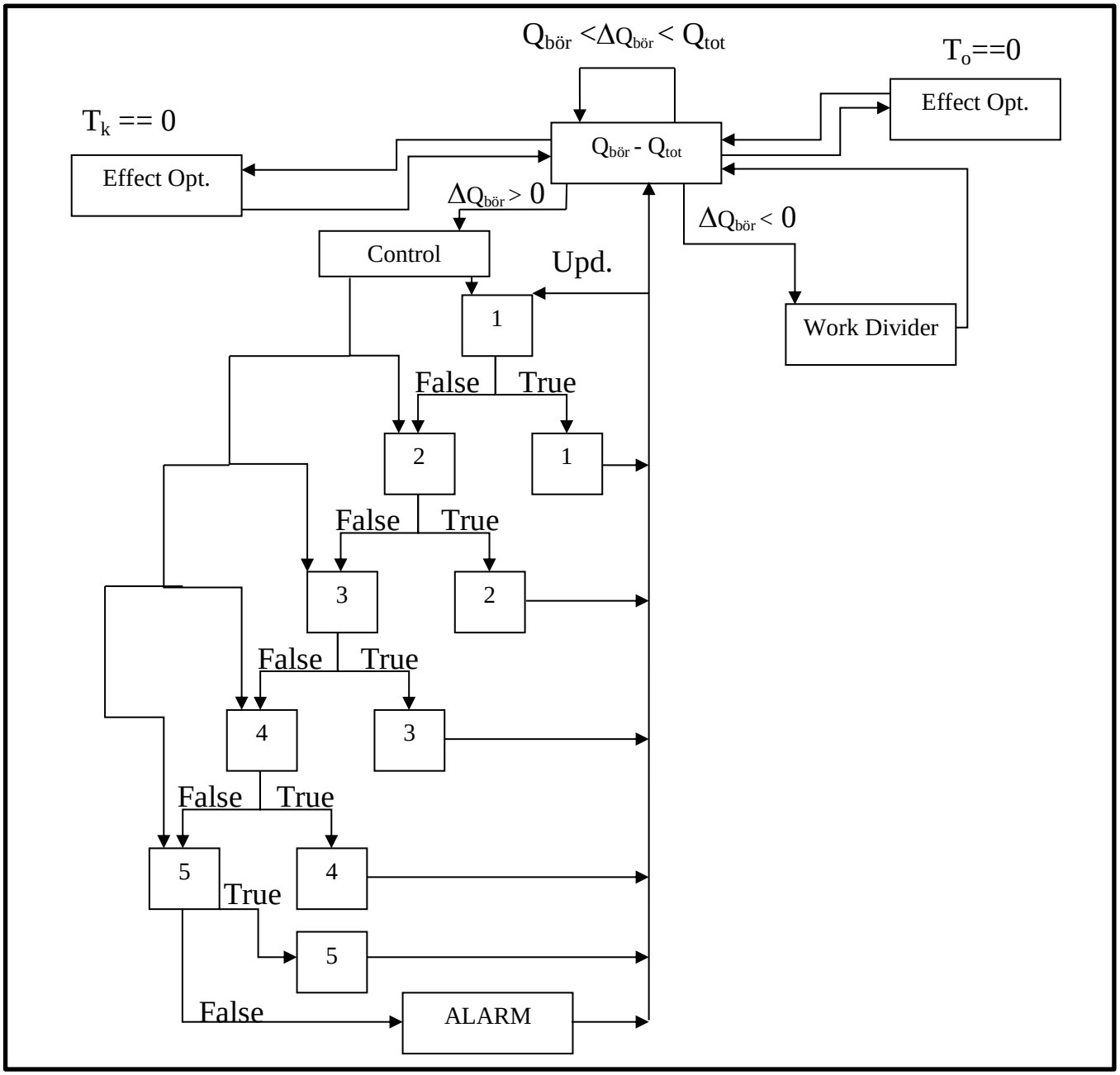
Såhär funkar systemets sökning efter lämpliga motorer, den uppfyller en del av våra viktiga mål, så som att den låter så få motorer som möjligt att köra och de motorer som körs ligger på sitt nominella värde, överdimensionering undviks väldigt bra. Jämför vi även med tidigare lösningsförslagen i analysen, är denna lösning implementations vänligast, då metoden kan tänka sig funka som en matris som styrs av två for-loopar och jämför hela tiden om kravet är TRUE.

Däremot håller inte denna lösning särskilt länge trots dem stora fördelarna, på grund av att vi inte uppfyller målet med att stänga av eller starta motorer så sällan som möjligt. Denna nackdel är väldigt stor, då detta skulle medverka till att trycket inte alltid kan hållas konstant som i sin tur ger för svagt eller ingen stråle alls under en vattenskärningsmaskins arbetsprocess.

4.10.7 Flow scheme version 2.0 (Variable Divider)

Eftersom vi var så nära med att uppnå alla målen i föregående lösning, bestämde vi oss för att fortsätta analysera och utveckla vidare på det tidigare flödesschemat efter att vi haft möte med vår handledare och hans kollega, som uppmärksammade just dem stora nackdelarna med det förra. Vi fick även idéer och tips från dem. Genom att använda tidigare idéer och kombinera med det gamla flödesschemat, kunde vi åstadkomma en komplett lösning som

uppfyller alla krav och mål. Det målet vi tog störst hänsyn till under utvecklingen var just hur vi ska göra för att stänga av och sätta på motorer så sällan som möjligt.



Figur 4.28. Arkitektur över algortimen med modifieringar.

Som vi kan notera, har olika nya block lagts till och någon har ändrats, det nya schemat kommer att förklaras helt, men vad vi kan ha i åtanke är att systemet fungerar fortfarande likt den förra med matris metoden.

Det centrala blocket som subtraherar börvärdet och det totala nuvarande flödet i systemet är identisk med det tidigare flödesschemat.

Metoden som används för att beräkna fram vilken av motorerna och hur många som ska startas vid ett angivet bördvärde fungerar också i princip på samma sätt fortfarande, men skillnaden är att en kontroll kommer göras före

för att kolla hur många motorer som redan är påslagna om ett nytt börvärde skulle vara så pass högre än det gamla värdet att man är i behov av att starta en till motor. Då kan kontrollen hänvisa en direkt till blocket som räknar om två motorer kan hantera det önskade börvärdet, om en motor var igång sedan tidigare. Istället för att passera genom block ett igen varje gång systemet kräver addering av en ny motor, man kan säga att den fungerar som en genväg i flödesschemat.

En stor skillnad på det nya och gamla systemet är som man kan notera att blocket *Shutdown* har byts ut till *Workdivider*, anledningen till bytet var dels för att vi måste ta hänsyn till målet vi nämnde innan, med att stänga och starta motorer så sällan som möjligt. Eftersom regelbunden simulation gjordes med matrismetoden för att kontrollera om systemet hade fått ett så pass högt börvärde att om man hade möjligheten att stänga av en hel motor utan att kriteriet större eller lika med börvärdet bryts. Detta bidrog till att så fort simulationen gav klart tecken för avstängning av en eller flera motorer så verkställdes detta och stängde av, startade eventuellt efter nytt börvärde, för ofta. Och som vi nämnde innan var detta en stor nackdel, som kunde fördröja ett högre börvärdes krav för att trycket ska höjas eller vara konstant, då motorerna kräver en viss tid till att startas. Arbetsdivideraren(Work Divider) fungerar som vi nämnt tidigare i analysen, att en jämn fördelning av arbetslasten görs mellan motorerna, dock så är skillnaden att den fördelar på dem motorerna som **endast** är igång. Work Divider blocket initieras när börvärdet sjunker och istället för att stänga av någon redan påslagen motor, dividerar vi istället arbetslasten. Vi har tagit väldigt mycket hänsyn till just fall där motorerna har olika maxkapaciteter och därför kommer en motor som ej kan hantera den jämnt dividerade arbetsfördelningen, precis som innan förklarat att köra på sin maxkapacitet, det som resterar kommer att adderas till arbetslasten som ska fördelas jämt en andra gång, men med en division med en motor mindre.

Effect optimization är två nya block som lagt till i flödesschemat, både blocken har samma funktion precis som dem har samma namn, men den ena initieras efter en konstant tid, T_k, medan den andra initieras efter en mindre bestämd tid, T_o, och den är valfri att användas. Effekt blockens uppgift är att försöka energi optimera på bästa sätt och undvika att någon av motorerna jobbar överdimensionering, vid ett börvärde som varar mer eller mindre en viss tid. Blockens funktion är den samma som vår tidigare *Shutdown* block, men inträffar nu istället vid två tillfällen och mer sällan än tidigare. När en motor ska stängas av, kontrolleras den minsta motorn först om den kan stängas av helt eller kan ersättas med en mindre motor, beroende på om bördesvärdes villkoret uppfylls eller inte, som illustrerades tidigare stegvist med matris

metoden. På så sätt kör vi på så få motorer som möjligt och vi undviker överdimensionering med.

Alarm blocket aktiveras endast om alla drivenheter finns i beräkningen, men kan fortfarande inte nå det önskade börvärdet. Då ska detta visas på displayen som ett sorts alarm och alla motorer sättas igång till att arbeta i 100 procent.

Nedan illustreras ett praktiskt exempel:

$Q_{1,max} = 100$			
$Q_{2,max} = 80$			
$Q_{3,max} = 60$			
$Q_{4,max} = 40$			
$Q_{5,max} = 20$			
Antal motorer	1	2	3

Låt oss säga att vi får ett flödesbörvärde på 200 liter/minut, då kommer systemet att kolla kontrollen om någon eller några motorer redan är igång och jobbar, eftersom vi tänker oss att exemplet illustreras från starten, vet vi att inget är påslaget. Kontrollen kommer därför att hänvisa till block ett där vi kontrollerar om en motor klarar av hela arbetsbördan själv, precis som tidigare visat.

$Q_{1,max} = 100$	1		
$Q_{2,max} = 80$	0		
$Q_{3,max} = 60$	0		
$Q_{4,max} = 40$	0		
$Q_{5,max} = 20$	0		
Antal motorer	1	2	3

Vi lägger till en etta på den sista störta vi kontrollerat, även om villkoret inte är ”TRUE” än. Sedan testar vi två motorer i kolumnen bredvid som vanligt.

$Q_{1,max} = 100$	1		
$Q_{2,max} = 80$	0	1	
$Q_{3,max} = 60$	0	0	
$Q_{4,max} = 40$	0	0	
$Q_{5,max} = 20$	0	0	
Antal motorer	1	2	3

Samma sak här som i föregående steg. Vi testar tre motorer i nästa steg.

$Q_{1,max} = 100$	1		
$Q_{2,max} = 80$	0	1	
$Q_{3,max} = 60$	0	0	
$Q_{4,max} = 40$	0	0	
$Q_{5,max} = 20$	0	0	1
Antal motorer	1	2	3

Nu har vi hittat tre lämpliga motorer som kan hantera vårt börvärde, men om börvärdet skulle sjunka till 180 liter/minuten skulle vi i tidigare fall göra en ny simulering och stänga av Q_2 och $Q_{5,}$ starta Q_3 för att få lämpliga motorer att köra ut 160 liter per minut. Istället kommer ingen motor att stängas av utan när det sker ett tryckfall, kommer en jämn fördelning att ske mellan motorerna, genom att ta totala flödet $Q_{bör}$ dividerat på antal motorer som är påslagna. I vårt fall blir beräkningen som följande:

$$\frac{160}{3} = 53.33 \text{ liter/minut per varje}$$

En sak vi kan observera är att båda motorerna Q_1 och Q_2 klarar av fördelningen, medans motorn Q_5 är för liten motor och klarar ej av att påverka sin pump till att höja flödet till cirka 53 liter/minut. Vi kommer ta en liknande åtgärd som vi gjorde i den första Work Divider funktionen som vi presenterade i början av analysen och den går till som följande:

Q_5 motorn kommer att köra på sitt Q_{max} och det resterande av de 53.33 literna kommer man att subtrahera med $Q_{5,max}$ det vill säga 20. Den kvarstående summan blir (53.33 – 20=33.33), 33.33 liter/minut och detta värde divideras sedan med dem antal motorerna som är kvar, $\frac{33.33}{2}$ =16.665 liter/minut och sedan adderas denna mängd till den första kvoten man fick (53.33+16.665= 69,995).

Resultatet av vår addition kommer att tilldelas till våra båda körande motorer och äntligen har vi nu löst fördelningsproblemet, nackdelen är dock att det finns en chans att överdimensionering kan uppstå för någon eller några motorer i en kort tidsperiod. Som vi förklarade innan kan vi även lösa överdimensionering med vår ”Effect opt” –block, som kan initieras antigen genom en längre bestämd konstant tid eller en mindre bestämd tid med valfrihet. När i vårt fall denna tid gått igenom och initierar funktionen i blocket, försöker funktionen att simulera bättre valmöjligheter att fortsätta

köra vårt nuvarande börvärde på 160 liter/minut som nu stått kvar en viss konstant tid. Resultatet bör vara fixandet av överdimensionering och om möjligheten finns att köra på få än tre motorer som vi i nuvarande läge har igång.
Vi kan direkt se att vi har en motor med ett $Q_{1,max} = 100$ och $Q_{3,max} = 60$ som ser ut att vara det bästa alternativet att gå efter och innan vårt system modifierar någon körning eller lägger till kommer en simulering att äga rum och komma fram till detta i matrismetoden inom ”effect opt.” –blocket .

$Q_{1,max} = 100$	1		
$Q_{2,max} = 80$	0	1	
$Q_{3,max} = 60$	0	0	
$Q_{4,max} = 40$	0	0	
$Q_{5,max} = 20$	0	0	1
Antal motorer	1	2	3

Först testas om den minsta motorn kan tas bort utan att villkoret blir ”FALSE” som vanligt. Och eftersom $Q_{tot} – Q_{5,max} >= Q_{bör}$ vilket blir (200-20 = 180) och 180 ger ”TRUE” då 180 >=160, då vet vi att vi kan förkasta hela kolumn tre, klara oss på endast två motorer. Samma test görs i kolumn två om en motor kan klara hela lasten, (180-80=100), 100 ger ”FALSE” då 100 >= 160 bryter villkoret, vi stannar då kvar i kolumn två och försöker hitta en mindre motor i samma kolumn tills vi hamnar på ”FALSE” igen, då kommer vi gå tillbaka till senaste ”TRUE” i andra kolumnen som följande visar:

$Q_{1,max} = 100$	1		
$Q_{2,max} = 80$	0	0	
$Q_{3,max} = 60$	0	0	
$Q_{4,max} = 40$	0	1	
$Q_{5,max} = 20$	0	0	0
Antal motorer	1	2	3

När vi väl får ”TRUE” på villkoret och hamnar på Q_4, får vi ”FALSE” på villkoret, eftersom ($Q_1+Q_4=140$) , 140 >=160 blir ”FALSE”, därför går vi tillbaka med ett sammarbeta mellan den tidigare testade det vill säga ($Q_1+Q_3 = 160$) , som vi kan se blir 160 >=160 ”TRUE” och därför verkställer vi detta och låter dessa två motorer köra vidare tills ett nytt börvärde kommer som kan göra att vi behöver ta olika åtgärder igen.

4.11 PID-Regulator och Reglerfel

Diskuterat framtill nu har handlat mest om börvärdet, men inte nämnt hur är-värde. drivenheterna som är tänkt att användas i systemet, innehåller PID regulatorer. Vi börjar med att beskriva varje del i regulatorn.

1. P-reglering: *Reglering där variationer i styrsignalgen Δu är proportionella mot felet. Styrsignalen kan därför anta godtyckligt värde inom ett givet intervall. Stort K-värde ger ett snabbt system som har dålig stabilitet. Litet K-värde ger bättre stabilitet men gör systemet onödigt långsam.* [7]

2. PID-reglering: *Integrerande verkan används för att eliminera kvarstående fel vid störningar och börvärdes ändringar.* (Referens: Modern Reglerteknik, sida 66) *Deriverande verkan används för att förbättra stabiliteten och/eller snabbheten. Deriverande verkan används aldrig ensam.* [7]

Ifall ett reglerfel sker, hur ska man hantera det då? Hur ska Mastern veta att Slavarna som kör har fått ett reglerfel och måste kompensera? Mastern måste veta hur mycket en Slav arbetar, genom att Slaven skickar sitt $Q_{är}$ kontinuerligt. Genom denna information, kan man säkerställa två saker. Det första är att Slaven verkligen arbetar och att kommunikationen emellan verkligen fungerar. Samtidigt kan Slaven kontrollera om Mastern verkligen finns där, genom att låta Mastern skicka ett acknowledgement för varje paket Slaven skickar.
Mastern måste kontrollera att det eftersträvade $Q_{bör}$ verkligen stämmer överens med $Q_{tot,ärvärde}$. Om ett regler fel uppstår måste man åtgärda det. Reglerfelet kan kalkyleras ut genom formeln:

$$Q_{tot,\,fel} = Q_{tot,\,bör} - Q_{tot,\,är}$$

Om det finns ett reglerfel, hur ska man åtgärda detta utan allt för stor tidsfördröjning? Mastern kan använda villkoret att om ett regler fel uppstår, får Slaven med minsta IP adressen ta hand reglerfelet och kompensera den. Men som vi har tidigare har nämnt är vårt totala flöde i systemet dynamisk, den ändrar sig kontinuerligt. Ändringen av totala flödesvärdet kan variera, förändringar av värdet kan ske i millisekunder upp till minuter.

Är det nödvändigt i så fall att ta hänsyn till reglerfelet, om vi har ett dynamisk flödesvärde för systemet? Nej, det är det inte, eftersom vi inte får glömma att det tar också en viss tid för att beräkna reglerfelet och därefter låta någon av drivenheterna att kompensera felet. Medan Mastern gör detta, kan man

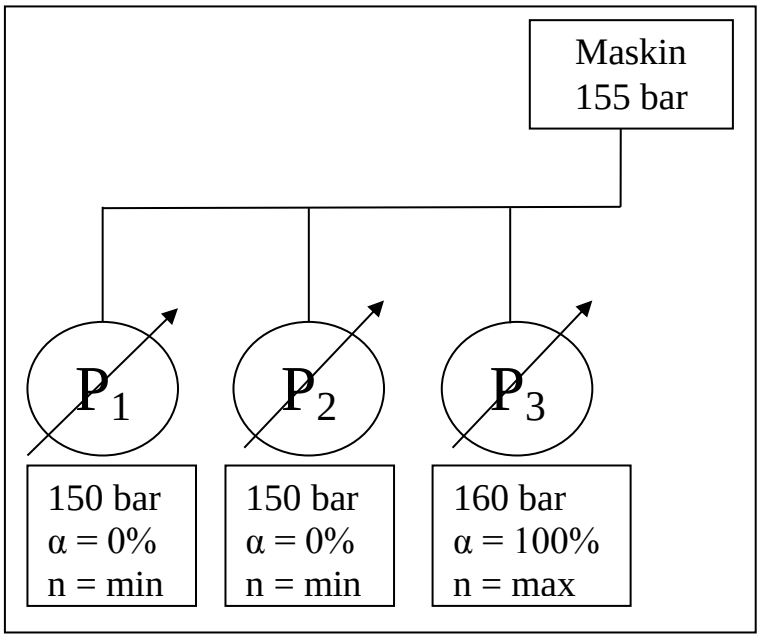
samtidigt få in ett nytt total flödesvärde för systemet, vilket gör det beräknade reglerfelet irrelevant, då man får ett nytt teoretisk reglerfel.

Hur löser vi detta problem? Det enklaste lösningen är helt enkelt att ignorera detta, eftersom förändringar i systemet sker i snabba intervall, vilket för att beräkning och kompensering av reglerfel inte är relevant i detta sammanhang då alla drivenheter är utrustade med PID-regulatorer, och kan eventuellt reglera sitt egna reglerfel. Vi tar endast hänsyn till det totala flödet.

4.12 Justerskruv för Tryckavskärning

Något som vi inte har nämnt är justerskruven för den variabla pumpen. Denna justerskruv gör det möjligt att justera trycket för den variabla pumpen. Problemet med detta är att detta är ett mekaniskt problem och inte något automationsproblem. För att förenkla förståelsen och förklara problematiken, illustrerar vi detta.

Låt oss säga att vår maskin eftersträvar ett tryck på 155 bar. För någon okänd anledning, har maskinoperatören skruvat på justerskruven hos pump nummer tre, vilket har gjort att pumpen vill uppnå ett tryck på 160 bar. Problemet med detta är att pump tre kommer **aldrig** att uppnå ett tryck på 160 bar, då pumpen inte kan nå detta trycket. Pumparna nummer ett och två kommer att vinkla sig tills det har uppnått sitt tryck och sedan vinkla sig tillbaka. Medan pump nummer tre aldrig kommer vinkla tillbaka sig.



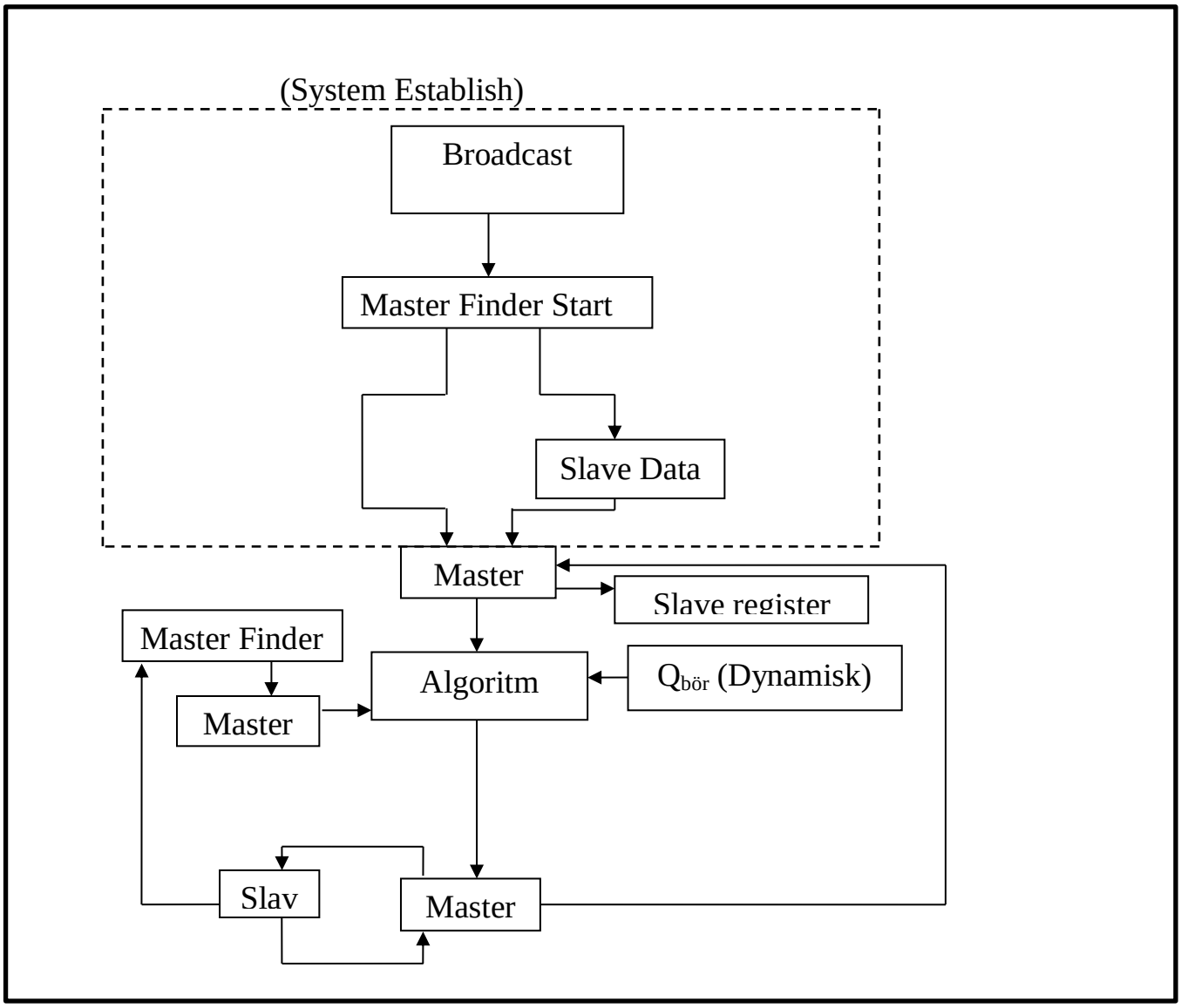
Figur 4.29. Förtydling av problematiken.

I dagsläget kan vi inte lösa problemet, då detta är ett mekaniskt problem. Det finns förnärvarande inget automatiserad system för detta. Vi kan endast ge våra rekommendationer. Maskinoperatören får helt enkelt sträva efter att skruva så sällan som möjligt på justerskruven för tryckavskärning. Annars kan detta problem uppstå om man ändrar positioneringen på skruven alltför ofta.

5 Resultat

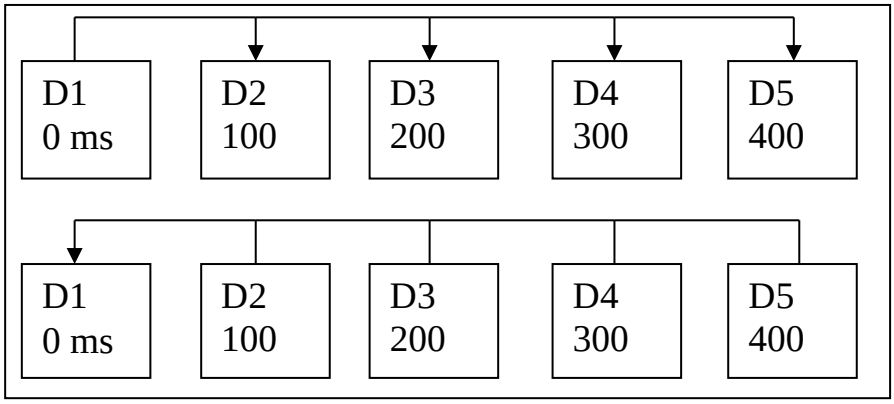
5.1 System för Master/Slave

Efter granskning av handledaren har resultatet av arbetet visat sig vara ett i teorin fungerande system. En sak som är värt att notera i detta stycke är att alla våra analytiska undersökningar inte hade relativ användning till vårt teoretiska system. Vi börjar med att repetera grundstrukturen för systemet stegvis och hur det är tänkt att exekveras stegvist, det vill säga att vi ska förklara processen för varje block och vilken data som ska skickas. I dagsläget har vi ännu ingen prototyp av själva systemet och därför förlitar vi oss på våra teorietiska beräkningar, företagets bekräftelse, samt våra referenser ur böcker och kursmaterial. Vi har strukturerat om vår grundstruktur jämfört med tidigare struktur i den analytiska delen. Skillnaden mellan den nya strukturen och den analytiska strukturen är att vi undviker ytligare tidsfördröjningar genom att vi tillåter mastern uppdatera slavens register med sin data.



Figur 5.01. Skiss över det teoretiska systemet.

Broadcasting: Innan en drivenhet startas, förprogrammeras en tidsfördröjning för varje drivenhet, för att på så sätt låta dem registreras i sekvenser för att få dem i en ordning från minsta IP-adress till den sista anslutna drivenheten i systemet. För varje drivenhet ökas tidsfördröjningen med 100ms och detta ger oss en linjär ökning för tidsfördröjningen.



Figur 5.02. Initiering för anslutning mellan drivenheterna.

Master Finder Start: I det här blocket kontrolleras först och främst om vi redan har en master i vårt system. Drivenheterna gör en förfrågan i systemet om det finns en master för tillfället, om det finns måste mastern i systemet returnera ett ”ja”, ställer sig den frågande drivenheten som slav. Om det istället skulle visa sig att efter en förfrågning, att det inte finns någon master i systemet, då ska drivenheten med minsta IP-adressen utses till master medans resten ska utses till slavar.

Slave Data: Efter att rollen har blivit utsedda, måste dem som blivit utsedda till slavar, skicka data angående deras motorkapacitet till mastern. Med detta menar vi informationen om maxflödet Q_{max}, beräknas som tidigare nämnt genom $Q_{max} = n_{max} * V * \eta$. Sedan skickas denna information vidare till den valda mastern.

Master Case One: Den utvalda mastern kommer vänta in att ta emot alla slavars ovannämnda data. När all data har skickats till mastern från alla slavar, kommer mastern att uppdatera alla slavars registerlista för att hålla reda på anslutna drivenheter, sedan går den vidare till algoritm blocket.

Slave Register Update: mastern uppdaterar slavarnas register med informationen om respektive drivenhets IP-adress och data.

Algorithm: När alla roller är klarsatta, är det dags att gå vidare till nästa steg som är algoritm blocket, mastern kommer då att initiera detta block som kommer att räkna och fördela ut flödes börvärdet till respektive drivenheter. Men innan någon arbetsfördelning tilldelas, kommer Algoritmblocket att få in vårt dynamiska totalflöde Q_{tot}. Börvärdet $Q_{bör}$ kan då räknas ut och sedan skickas till undre masterblocket som kommer sedan att tilldela alla drivenheter sitt $Q_{bör}$.

Master Case Two: Den undre masterblocket skickar ut $Q_{bör}$ som den fått ta emot av algoritmblocket, dessa $Q_{bör}$ som kan vara både lika och olika mängder, skickas från masterblocket till respektive slav. Om en slav skulle försvinna, kan detta upptäckas genom masterns cykliska tid, som kontrollerar är-värdet, denna tid motsvarar 1ms.

Slave: Efter att slavarna varje gång tar emot ett $Q_{bör}$ från mastern, skickat dem tillbaka ärvärden. Detta ärvärdet består av flödet $Q_{är}$, varvtalet $n_{är}$ och trycket $p_{är.}$ Om mastern skulle försvinna, skulle detta upptäckas genom ”time for new master”(TFNM) initieras efter en viss tidsfördröjning. TFNM är som vi nämnde tidigare, börjar om med en nedräkning efter varje gång den mottagit ett paket från mastern. Skulle kravet för TFNM nu gå igenom, kommer slavarna initiera ”Master Finder”-blocket.

Master Finder: I detta block kommer nu mastern som försvann att raderas ur registret och sedan kommer blocket att utse en ny master, genom att kolla efter drivenheten med lägst IP-adress i registret. Resterande fortsätter som slavar, sedan fortsätter processen till algoritmblocket, där en eventuellt ny arbetsfördelnings räknas ut.

Fall som kan uppstå som vi tidigare nämnt som t.ex. om det skulle finnas två masters, löser vi beroende på situationen som vi nämnde under rubriken ”New driver”. Man kan läsa mer igen om en master respektive slavs fall om någon av dem skulle försvinna i rubrikerna ”master gone” och ”slave gone”.

Som tidigare nämnt har vi ingen färdig prototyp, därför tar vi ett exempel hur systemet är tänkt att fungera ur teoretiskt perspektiv.

1.Om vi har t.ex. fem drivenheter i vårt subnät och har IP-adresser med dessa nummer på slutet:

(01, 02, 03, 04, 05). Alla drivenheters motorer i detta fall har fyrpoliga motorer, samma verkningsgraden och deplacementet i systemet. Som vi nämnt tidigare har vi en tidsfördröjning mellan varje drivenhet på 100ms. Det vill

säga att broadcasta och registrera alla anslutna drivenheter i registret tar cirka 400ms.

2. Därefter utser drivenheterna sina roller beroende på drivenhetens IP-adress, i detta fall har drivenheten med 01 i slutat av IP-adressen, utser sig till systemets master, eftersom den innehar lägst IP-adress enligt registret, medans dem andra utser sig själva till slavar. Mastern kommer sedan att invänta data från alla slavar och går inte vidare till algoritmen förrän alla slavar har skickat.

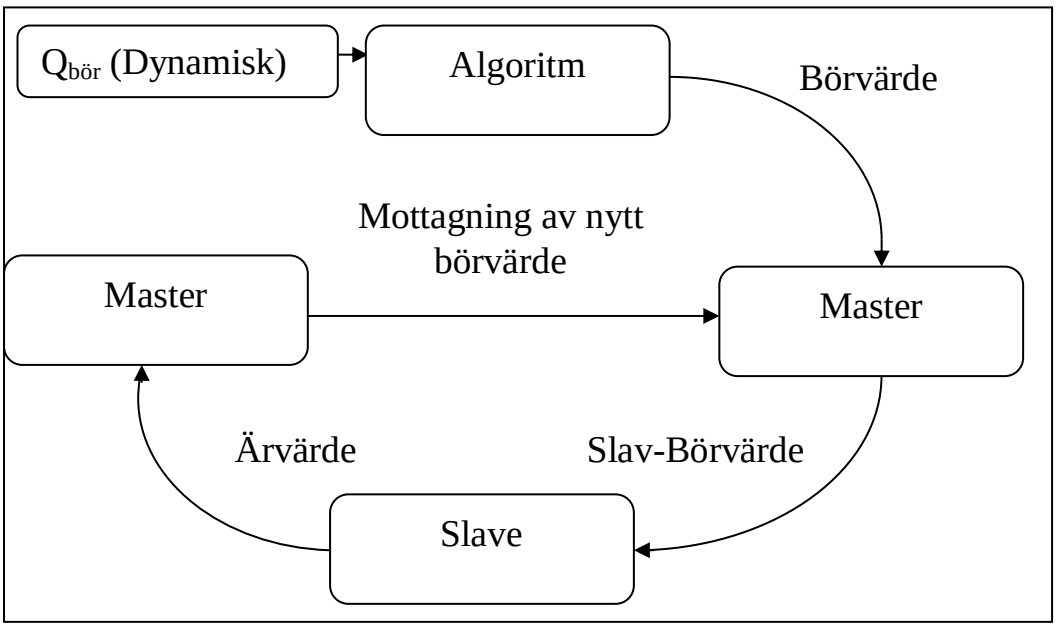
3. Data som slavarna(02, 03, 04, 05) kommer att skicka är $Q_{max} = n_{max} * cm^3 * \eta$, beräknas av respektive slav.

4. Master kommer nu att uppdatera alla slavars register om varandras Q_{max}.

5. När master nu tagit emot all beräknad data, registreras dessa data mängder för respektive drivenhet. Sedan initierar mastern algoritmfunktionen. Algoritmen kommer att få ett dynamiskt Q_{tot} som den kommer att fördela bland alla fem drivenheter och denna fördelning blir vår $Q_{bör,}$ detta meddelas till mastern som kommer att skicka vidare $Q_{bör}$ till respektive slav.

6. Slavarna tar emot sitt $Q_{\alpha,\ bör}$ och returnerar i sin tur sitt $Q_{är}$ till mastern.

7. Nu befinner sig systemet i underhållnings loop, där algoritmen får kontinuerligt nya $Q_{bör}$ värden och beräknar ut $Q_{\alpha,\ bör}$ till mastern som sedan skickar detta till slavarna som tidigare nämnt. Bilden illustrerar hur detta kan tänkas se ut (Figur 5.03).

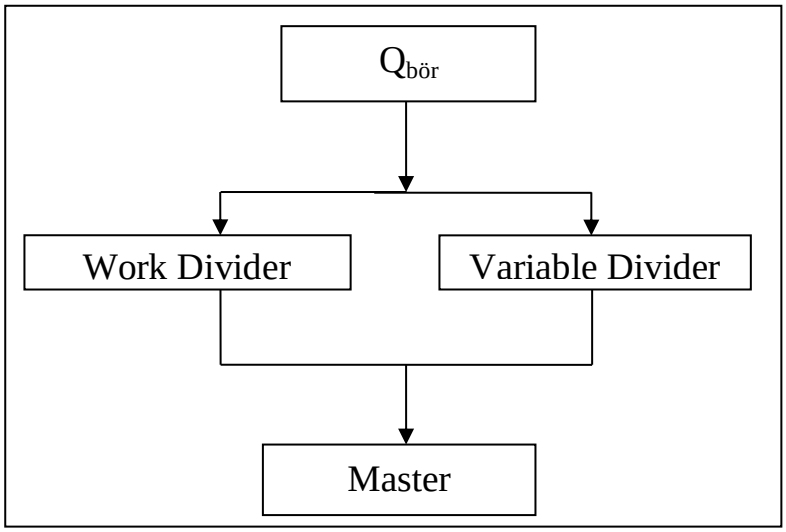


Figur 5.03. System underhåll.

Vårt huvudmål i projektet var främst att energieffektivisera genom att fördela arbetslasten mellan motorerna. Enligt vår undersökning på flera olika lösningsmetoder, kom vi fram till att av alla förslag och idéer vi hade, var dem bästa, mest relevanta lösningarna både WD samt selektivitet med kombinatoriskt test.

5.2 Algoritm (Work Divider och Variable Divider)

I vår analytiska del kan man observera vår tänkbara struktur för algoritmen, och förslag vi hade, dock ser resultatet annorlunda ut jämfört med den tidigare strukturen. Varför vi valt denna som vårt slutgiltiga resultat kommer vi förklara i följande textsnitt, medans längre fram i diskussionsdelen, kommer vi att förklara nackdelarna på vårt första förslag på strukturen.



Figur 5.04. Algortimens valmöjligheter.

Den här strukturen som vi beskriver här är den designen vi har tänkt använda oss av i algoritmen. Tanken är dock nästan detsamma, man mottar ett dynamisk flödesvärde för systemet, därefter kan man gå två vägar beroende på vilken optimering maskinoperatören strävar efter, antingen vidare till Variable Divider eller WD funktionen. Kunden eftersträvar egentligen efter att **alla** motorer har en jämn arbetsfördelning, men på begäran av examenshandledare har vi också undersökt när det är lämpligt att nollställa arbetslasten för respektive motor.

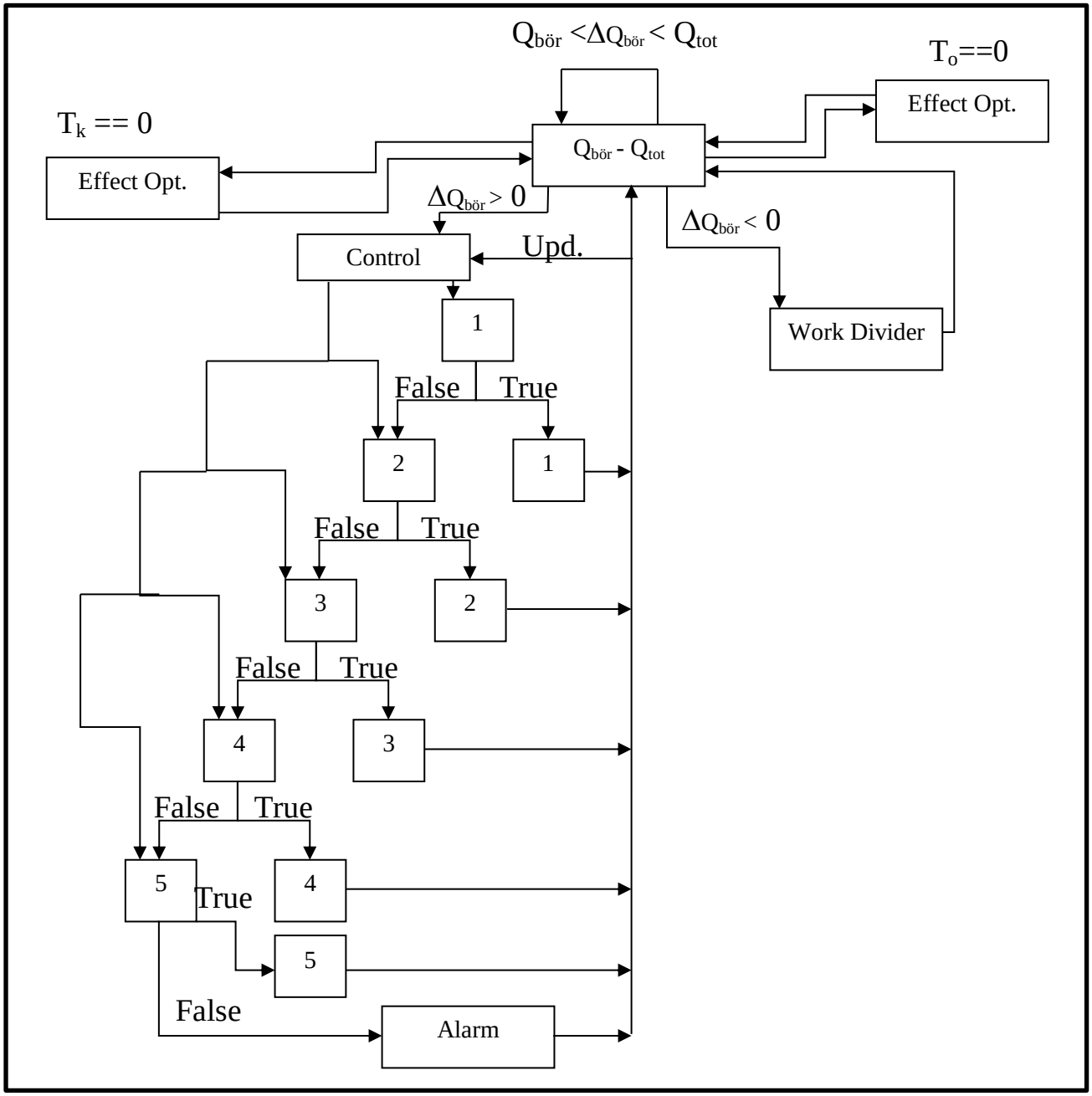
Work divider (WD) använder sig av simpel mattematik, genom använda den nu kända formeln:

$$Q_{\alpha,\ bör} = \frac{Q_{tot}}{D_n}$$

WD kommer alltid att endast fördela arbetsbelastningen jämt mellan alla drivenheter.

Energieffektiviseringen skall kunna ske oberoende på om motorerna har lika eller olika motorstorlekar, därför introducerar vi **Variable Divider**, där tanken är att den ska kunna fördela arbetet mer energieffektivt. Mastern har informationen om sin och alla slavars maximalla flödeskapacitet, med detta kan den simulera fram vilka motorer som är lämpliga att sättas i drift. Detta ska kunna ske samtidigt som våra primära mål uppfylls.
Den stora skillnaden mellan WD och Variable Divider funktionerna, är att WD fördelar arbetslasten jämnt emellan alla drivenheter, medan Variable Divider har möjligheten att stänga och starta motorer. Variable Divider funkar på sådan sätt att den är mer intelligent och tar olika åtgärder beroende på vilka händelser som sker. Variable Divider baseras på matris-metoden, sid 59 (Flow scheme version 2.0 (Variable Divider)), där den simulerar fram de motorer som endast behövs för att möta flödesbehovet.

För att undvika problematiken med överdimensionering, introduceras två nya block med uppgift att energieffektivisera systemet, där en tid, T_k, och en annan valfri tid, som är kortare, T_o. Båda tiderna bestäms av maskinoperatören. Tanken är att så länge det inte sker en börvärdesändring, kommer tiden att räknas ner, om det sker en ändring av börvärdet, startas nedräkningen om. Skulle det ske att nedräkningen når noll, är det en indikation på att systemet **kan** köras mer energieffektivt, och blocket initieras. När initieringen sker, aktiveras matris-metoden igen, fast utan hänsyn till vilka maskiner som är aktiva innan, utan endast hänsyn till att ha så få motorer som möjligt, nära sin nominella värde. Alarm blocket aktiveras endast om alla drivenheter finns i beräkningen, men kan fortfarande inte nå det önskade börvärdet. Då ska detta visas på displayen som ett sorts alarm och alla motorer sättas igång till att arbeta i 100 procent.



Figur 5.05. Struktur för algoritmen.

6 Diskussion

6.1 Slutsats

Vi lyckades uppnå i princip alla mål, förutom själva implementationen av systemet. Undvika överdimensionering så gott som möjligt, ha så få motorer i drift, uppnå även nominell drift till viss del. Med tanke på våra förstudier, kommer systemet i teorin ge en ljuddämpad arbetsmiljö.

6.2 Kommentarer

Förutom att de viktigaste målen uppnås, var vi tvungna att kompromissa med vissa mål, som överdimensionering. Då vårt system är tidskritiskt, kan vi inte stänga av och starta motorer hur som helst för att enbart undvika överdimensionering. Men samtidigt var vi fortfarande tvungna att tänka ut en sorts åtgärd mot denna sorts problem.

I Marginal kapitlet är det mycket som är odefinierad, vilket är en av orsakerna till att vi förkastade denna idéen, då det bidrog till alltför många frågetecken.

Work Divider funktionen i Algoritmen behövs egentligen inte, då vi använder variabla pumpar, kommer dessa att kompensera vid en eventuell tryckminskning.

Då detta system i dagsläget inte existerar, har vi sagt att implementationen sker endast i mån av tid, samtidigt att vi tar hänsyn till att göra implementationen så smidigt och mindre komplicerad som möjligt till de som ska implementera systemet.

Orsaken till att vi inte hann med implementationen, var att vi satsade mer på förundersökning, analys och strukturering av algoritmerna. Vårt huvudsyfte var att komma fram till **hur** man ska göra på bästa sätt för att uppnå de målen man vill med energieffektivisering. Samtidigt följde vi ett citat som återspeglar av vad vi har lärt oss.

”Tänk först, programmera sedan”
- Per Fouyer [8]

6.3 Utvecklingsmöjligheter

Som systemet är tänkt att se ut, kommer vi ha ett centraliserat system, där mastern är den som utför beräkningar, skickar/mottager information till slavarna. I framtiden kan man kanske utveckla ett decentraliserat system, där slavar kommunicerarar med varandra och utför egna beräkningar, där mastern enbart har i uppgift att motta information från andra slavarna. Skillnaden blir då att vi är mindre beroende av en master, samt mer självständighet för slavarna och de får fatta mer beslut.

Broadcasting algoritmen fick vi tips på att den kunde vara mindre överflödig med så många alternativ och åtgärder, dvs. att kanske ha få block som gör mer arbete istället i framtiden.

7 Referenser

7.1 Källkritik

Det var har varit svårt att hitta källor, då systemet inte existerar. Däremot användes kurslitteraturen *Power Electronics: Devices, Converters, Control and Applications*, vilket är en pålitlig källa asynkronmotorer och effektförluster och överdimensionering. Detta gällde även för källorna kring variabla pumpar, då denna information presenterades för oss av vår handledare. Källor till Profibus, var relativt enkla att hitta. Dock var det svårt att hitta källor som gav informationen om nackdelarna med systemet, Profibus, då de stora företagen, t.ex. Siemens, gav inte denna information.

7.2 Källor

[1]PE: Power Electronics: Devices, Converters, Control and Applications, M. Alaküla, P. Karlsson IEA/LTH 2006

[2]http://www.bosch-home.se/robert-bosch.html
(2014-04-28)

[3] http://www.automation.com/library/articles-white-papers/fieldbus-serial-bus-io-networks/introduction-to-profibus-dp
(2014-04-28)

[4] http://www.profibus.se/PDF/PA_System_Description_swe.pdf
(2014-04-28)

[5] http://www.feldbusse.de/Profibus/Buszugriffsprotokoll.shtml
(2014-04-28)

[6]http://www.boschrexroth.com/country_units/europe/sweden/sv/company/fairs/2014/InspirationBorlaenge2014/
(2014-05-04)

[7] Modern Reglerteknik av Bertil Thomas, sida 66, ISBN: 9789147093236

[8] Microprocessor Teknik av Per Foyer, sida 69, ISBN 9789144038766

8 Terminologi

TFNM
Time For New Master

OFC
Over Flow Control

LFC
Low Flow Control, första namnet till Selektivitet med kombinatorisk test

WD
Work Divider

VD
Variable Divider

WCS
Worst Case Scenario

PLC
Programmable Logic Controller