T E C H N I S C H E U N I V E R S I T E I T D E L F T

Vakgroep Telecommunicatie en Verkeersbegeleidingssystemen
Laboratorium voor Automatische Verkeerssystemen

Aard : Afstudeerverslag
Omvang : 57 bladzijden
Datum : januari 1987

Lab/Afd. : Telecommunicatie en Verkeersbegeleidingssystemen

Opdracht-
nummer : dK/1986/7

Auteur : M. Schoenmaker

Titel : de Besturing van Distributienetwerken
met name de besturing van het Richards netwerk.

Korte
inhoud : Er is een onderzoek gedaan naar algorithmes voor de besturing van (breedband) niet blokkerende omroep-netwerken. Het onderzoek heeft zich toegespitst op het beschrijven van de besturing van het `Richards'-netwerk. Hierbij is de `taal van Dijkstra' gebruikt.

T E C H N I S C H E U N I V E R S I T E I T D E L F T

FACULTEIT DER ELEKTROTECHNIEK
Vakgroep Telecommunicatie en Verkeersbegeleidingssystemen
Laboratorium voor Automatische Verkeerssystemen

a f s t u d e e r o p d r a c h t

Te verrichten door: M. SCHOENMAKER

De opdracht zal worden uitgevoerd bij A.P.T. Hilversum.

Mentoren: ir. J.C.A. Boekhorst (APT)
ir. R.A. Beukers (TU Delft)

Onderwerp: Besturing van het Richards netwerk.

Omschrijving van de opdracht:

De opdracht omvat een onderzoek naar algorithmes voor de besturing van (breedband) niet blokkerende omroepnetwerk. Dit onderzoek moet zich toespitsen op het onderzoek van een aantal aspecten van het zogenaamde "Richards'-netwerk (zie IEEE transactions on communications, Vol. COM-33, no. 10. oktober 1985, A two stage rearrangable broadcast switching network, door G.W. Richards em F.K. Hwang).

Aanvang opdracht: Opdrachtnummer: dK/1986/7

1 mei 1986 Delft, januari 1987,

DE BESTURING VAN DISTRIBUTIENETWERKEN,
met name de besturing van het Richards netwerk.

M.Schoenmaker
januari 1987
TU-Delft
Electrotechniek

VOORWOORD

Dit rapport is een verslag van het afstudeerwerk dat verricht werd bij APT-Hilversum. Het is een eerste onderzoek op het gebied van de besturing van distributienetwerken waarbij het tot nu toe meest effectieve netwerk, het Richards netwerk, als uitgangspunt wordt gebruikt.

Uitgaande van de resultaten van dit rapport kan met behulp van simulaties naar optimalisaties worden gezocht.

Ik ben APT in het algemeen veel dank verschuldigd voor het mogelijk maken van dit afstudeerwerk. En in het bijzonder mijn begeleider, Han Boekhorst, voor de vele discussies over het oplossen van bepaalde problemen.

INHOUDSOPGAVE

SAMENVATTING

Het doel van dit verslag is het geven van inzicht in de problematiek van distributienetwerken.
Op het gebied van dialoog verkeer is al veel bekend. Het Clos netwerk is het beste, tot nu toe bekende, netwerk voor dialoog verkeer (zoals telefoon), maar helaas niet geschikt voor het afhandelen van distributie verkeer.

Het Richards netwerk is het tot nu toe meest geschikte distributie netwerk. De blokkeringsvrije werking wordt slechts verkregen door de mogelijkheid om bestaande verbindingen tijdens hun houdtijd te verleggen.
Het onderzoek is toegespitst op het beschrijven van de besturing van Richards netwerken. Hierbij is de taal van Dijkstra gebruikt.
Om het aantal keren dat er verlegd moet worden te beperken bestaan er een aantal methoden, waarvan de twee belangrijkste zijn:

1 aanstampen, dat wil zeggen de verbindingen zo scheef mogelijk over het netwerk verdelen.
2 uniform leggen, dat wil zeggen voor elk station evenveel mogelijkheden vrijhouden

Aangezien met methode 1 net zulke goede resultaten te behalen zijn als met methode 2 en er voor methode 1 veel minder administratief werk moet worden gedaan om de momentele toestand van het netwerk bij te houden is gekozen voor methode 1.

Om tijdens het verleggen snel en eenvoudig een vrije mogelijkheid te vinden wordt hier ook bij de keuze van het nieuwe pad steeds de zwaarst belaste mogelijkheid gekozen waardoor de kans het grootst is dat het kortste pad wordt gevonden.

SUMMARY

The purpose of this report is to provide insight into the problem area of distribution networks.
A lot is known about dialogic traffic. The best network for dialogic (for instance telephone) traffic known to date is the Clos network, but unfortunately this network is not appropiate for handling distribution traffic.

The Richards network is the best distribution network known to date. To achieve a nonblocking operation, it may be required to rearrange existing paths.
The investigation is focussed on the control of Richards networks. To describe this control a language, introduced by Dijkstra is employed.

Several methods are known to limit the amount of rearranging, the two most important methods are :

1 "compacting", that is, select the possibility that already carries the largest traffic.
2 "spreading", that is, retaining the same number of possibities for each inlet channel.

Method 1 is favoured since it produces as good results as method 2 at reduced overhead.

The same method is also used to reduce the complexity of the rearrangement process.

INLEIDING

De komst van breedband applicaties stelt aan de communicatie centrales van morgen nieuwe eisen, die hun structuur ingrijpend beinvloeden.
Allereerst treedt er in het evenwicht tussen het gebruik van time switching en space switching technieken een sterke verschuiving op in de richting van space switching.
Voorts mogen deze centrales geen blijk geven van interne blokkeringstoestanden. Blokkeringsvrije centrales die gebruik maken van space switching technieken zijn reeds bekend voor spraakverkeer (Clos netwerken). Zij kunnen zelfs zo gedimensioneerd worden dat het voor de blokkeringsvrije werking vereist kan zijn om een verbinding gedurende zijn houdtijd een aantal malen van een nieuw intern pad te voorzien (rearrangeable Clos netwerken).

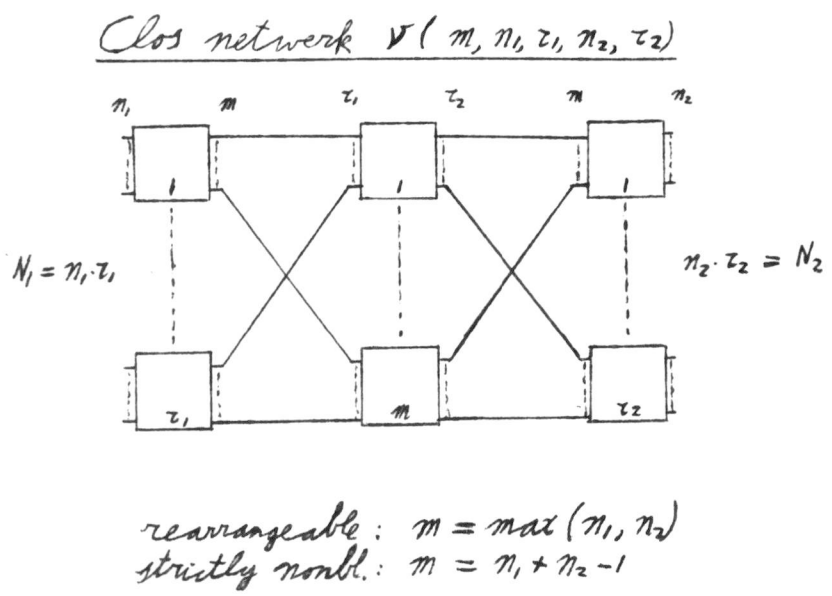
Helaas zijn Clos netwerken niet bruikbaar als blokkeringsvrije netwerken voor broadcast/distributie toepassingen (zoals kies-TV).
Er zijn space switching netwerken bekend die ook blokkeringsvrij zijn voor distributie toepassingen. Het meest effectieve van deze is het Richards netwerk. Ook bij dit netwerk wordt de gegarandeerd blokkeringsvrije werking slechts verkregen door verleggen van bestaande verbindingen.
Evenwel is nog onbekend welke bestaande verbindingen hiervoor in aanmerking komen en hoe dit verleggen het best kan geschieden.

Het doel van dit rapport is de besturing van dit Richards netwerk te onderzoeken en te beschrijven. Aangezien er nog niets bekend is op het gebied van de besturing van distributie netwerken zal eerst gekeken worden hoe het leggen en verleggen van oproepen/verbindingen op zich gaat. Daarna wordt nog onderzocht wat voor mogelijkheden er zijn om het aantal keren dat er verlegd moet worden te beperken en als er verlegd moet worden dit zo slim/snel mogelijk te doen.
De besturing wordt in drie programma's beschreven. Hierbij wordt er in het eerste programma van uitgegaan dat alle oproepen te leggen zijn. Vervolgens wordt in het tweede programma het verleggen toegevoegd. En tenslotte wordt in het derde programma gekeken hoe het leggen en verleggen zo slim/snel mogelijk gedaan kan worden.

In Hoofdstuk 1 wordt het gebied waarop het afstudeerwerk zich afspeelt, namelijk switching, besproken. Na in Hoofdstuk 2 de manier van programmeren toegelicht te hebben wordt in Hoofdstuk 3 het Richards netwerk besproken. In Hoofdstuk 4 worden vervolgens de drie programma's uitgewerkt. En tenslotte worden in Hoofdstuk 5 de conclusies van dit afstudeerwerk getrokken en worden er aanbevelingen gedaan hoe met dit onderwerp verder zou kunnen worden gegaan.

1.1 Dialoog verkeer

Op het gebied van dialoog verkeer is al veel bekend. In 1953 publiceerde Clos [1] een space switching netwerk structuur die nu nog steeds de beste is, althans voor symmetrische netwerken.
Hieronder volgt een schema van het Clos netwerk.



figuur 1

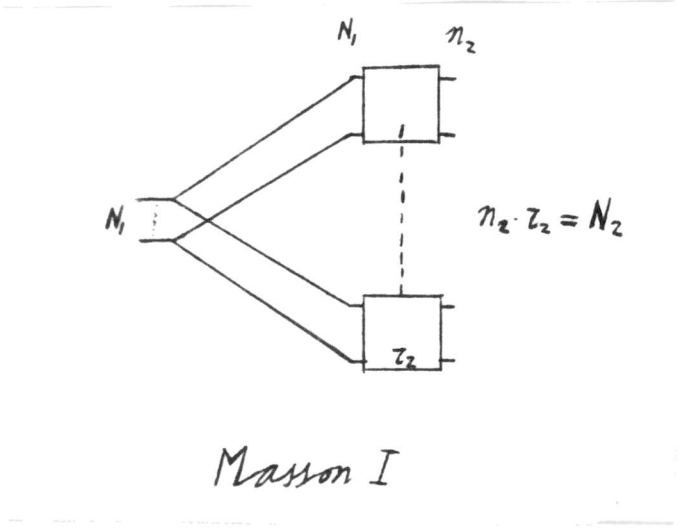
Dit netwerk kan zo gedimensioneerd worden dat het voor het niet blokkeren noodzakelijk is dat bestaande verbindingen tijdens hun houdtijd verlegd moeten kunnen worden (rearrangeable, m=max(n1,n2)). Dit verleggen moet geschieden zonder onderbreking van de service aan de abonnee's. Dit stelt hoge eisen aan de besturingstechnologie. Om onafhankelijk te zijn van de besturingssnelheid is het nodig dat een call verlegd kan worden zonder onderbreking (make before break). Door het verleggen nu sequentieel te doen hoeft de besturingshardware minder snel te zijn. Hiervoor is dan wel een extra middenswitch nodig.
In 1980 publiceerde Melas [2] een verleggingsprocedure waarbij de extra middenswitch geimplementeerd wordt door een tweetal bussen. Een oproep waarvoor het netwerk een blokkerende toestand heeft, kan meteen afgehandeld worden maar dan via een van de bussen en vervolgens worden in een procedure de bussen vrijgemaakt.

1.2 Distributie verkeer

Men wil in de toekomst de ether vrij maken voor mobiele communicatie. Hiervoor is het nodig dat distributieve diensten als radio en TV via een ander medium aangeboden moeten worden. Dat medium zal glasvezel worden, maar zelfs glasvezel heeft een transmissie begrenzing. Dit kan opgelost

worden door nu niet meer alle TV-kanalen naar de abonnee te sturen maar ca. vier die door de abonnee vrij gekozen moeten kunnen worden. Het schakelen gebeurt dan in de centrale. Het netwerk mag dan uiteraard geen blokkering vertonen.
Door Hwang en Jajszczyk [3] is geprobeerd om voor distributieverkeer het bekende Clos netwerk te gebruiken. Om het Clos netwerk blokkeringsvrij te maken voor distributieverkeer is een zodanige dimensionering nodig dat het aantal kruispunten groter is dan bij een noncomposiete (=niet in verschillende trappen verdeelde) implementatie. Clos netwerken zijn dus niet geschikt voor distributie verkeer.
In 1977 heeft Masson [4] een methode voor een betere netwerkstructuur bedacht. Deze methode kan onderverdeeld worden in drie stappen.

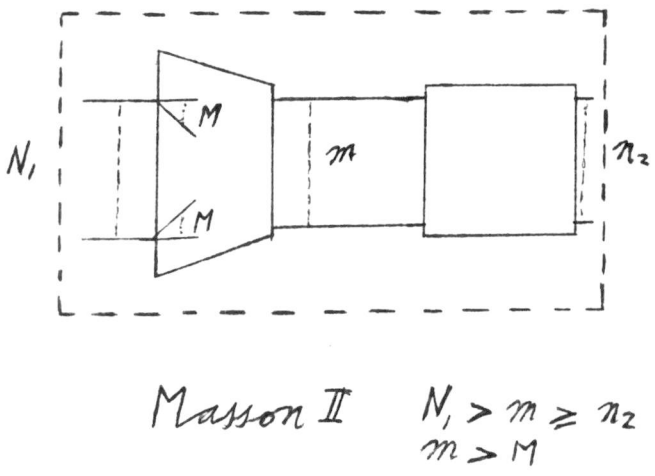
De winst - ten opzichte van de noncomposiete kosten N1*N2 - moet gezocht worden aan de kant van de stations (inputs). Het grote aantal outputs (abonnee's) kan dus bereikt worden door parallel-schakeling van een aantal identieke netwerken met N1 (dezelfde!) stations en n2 (steeds verschillende) outputs. Dit is de eerste stap van de methode van Masson : implementeer een distributie netwerk als een trap van concentrerende distributie netwerken. Als eerste trap is nu een connectie trap ontstaan (zie figuur 2).



figuur 2

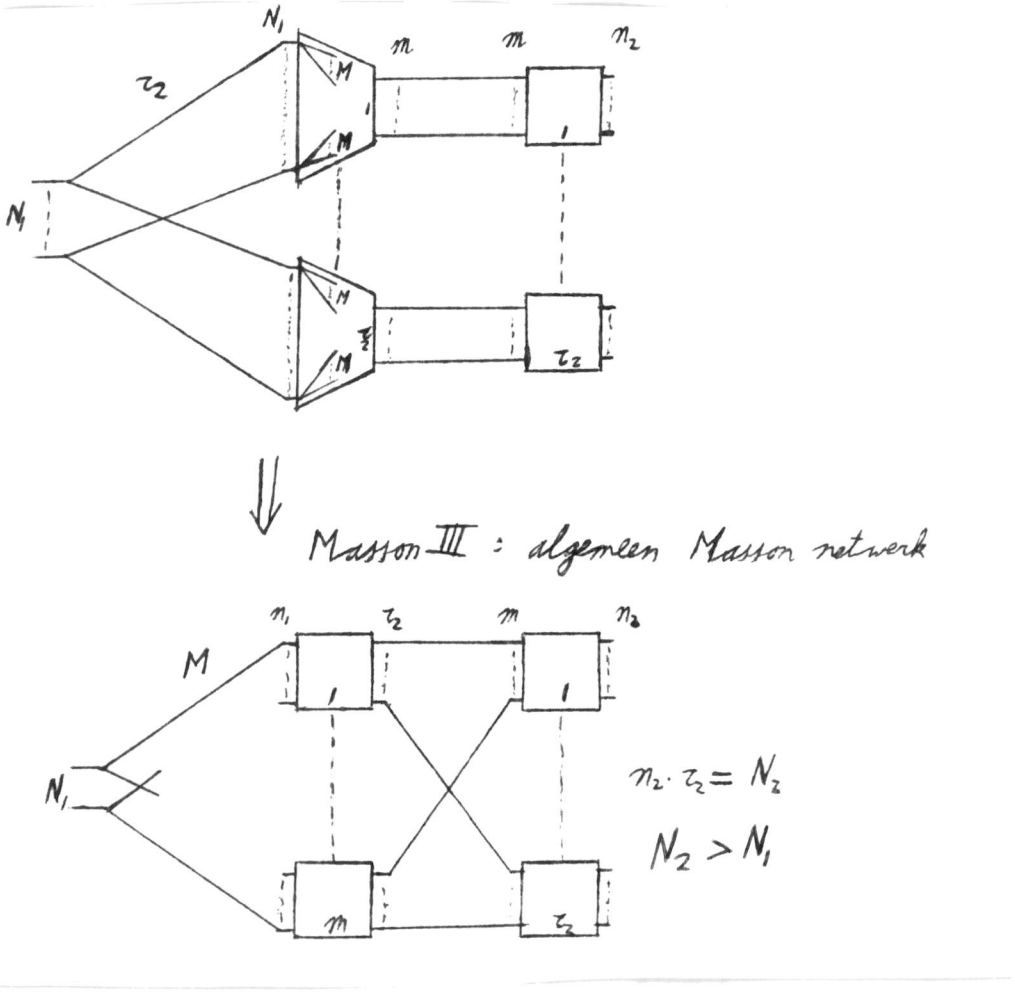
Het maken van een concentrerend distributie netwerk is de tweede stap van de methode: cascadering van een zgn. concentrator en wederom een concentrerend distributie netwerk (zie figuur 3). Een concentrator is een punt-punt netwerk terwijl een concentrerend distributie netwerk een distributie netwerk is. Iedere ingang van een concentrator kan slechts met M van de m (M < m) uitgangen worden verbonden (dit heten forks). Door deze recursie kan zoveel mogelijk van de schakelfuncties in de laatste switch worden gelegd. Dit heeft het voordeel dat, als er

meerdere abonnee's met hetzelfde station verbonden moeten worden, de distributie in deze laatste switch geschiedt. Er bestaan netwerken van dit type met minder kruispunten dan het Clos netwerk.



figuur 3

Een concentrator wordt geimplementeerd als een onvolledig van kruispunten voorziene noncomposiet. Dit betekent dat een gekozen station steeds via precies een kruispunt doorgeschakeld wordt.
De laatste stap van de methode bestaat uit het toepassen van een netwerk equivalentie: De forks in de eerste trap en de daarop volgende forks in de tweede trap worden in volgorde verwisseld. Het onvolledige karakter van de concentrators (stations M-voudig aanwezig) komt nu expliciet tot uiting in de connectie trap (zie figuur 4).



figuur 4

Netwerken opgebouwd volgens dit principe kunnen onderling verschillen door de keuze van de gebruikte concentrator. De door Masson zelf gebruikte concentrator, namelijk een binomiale concentrator, is verre van optimaal. Masson [10] noemt zijn concentrator optimaal, maar de randvoorwaarden zoals hij die stelt aan het aantal ingangen, het aantal uitgangen en aan de capaciteit zijn niet optimaal.
De tot nu toe meest geschikte concentrator is die welke gebruikt wordt in het Richards netwerk.

Het probleem van een extra middenswitch danwel bussen zoals bij dialoog verkeer doet zich bij distributie verkeer niet voor aangezien het daar altijd mogelijk is de nieuwe verbinding te leggen voordat de oude verbinding afgebroken wordt.

2 MANIER VAN PROGRAMMEREN: De taal van Dijkstra

Voordat aan de besturing van het distributie netwerk in Hoofdstuk 3 begonnen kan worden, zal eerst een keuze gemaakt moeten worden hoe de programma's gepresenteerd gaan worden.
In paragraaf 2.1 wordt de taal van Dijkstra besproken waarna in paragraaf 2.2 de keuze van deze taal wordt toegelicht.

2.1 Beschrijving van de taal van Dijkstra

Deze taal is heel eenvoudig van opbouw, waardoor het programmeren overzichtelijker kan blijven. Dijkstra [7] maakt bij zijn taal gebruik van "guarded commands" waarbij een "guarded command" als volgt te beschrijven is:

```
<statement list> ::= <statement> {;<statement list>}
<guarded command> ::= <boolean expression> -> <statement list>
<guarded command set> ::= <guarded command> {
                          []<guarded command>}
```

Het teken [] geeft de scheiding aan tussen de verschillende guarded commands (zie voorbeelden). Dijkstra noemt de boolean expressie een guard. Een guarded command is een bouwsteen voor een statement. Een statement wordt namelijk altijd uitgevoerd als het programma daar aangekomen is, terwijl bij een guarded command de boolean expressie 'true' moet zijn om uitgevoerd te worden.
De taal van Dijkstra biedt twee manieren om van guarded commands statements te maken.
De eerste manier is het if....fi paar

```
<statement> = if<guarded command set>fi
```

Dit statement kan leiden tot niet-determinisme, als namelijk meerdere guards 'true' zijn moet er een keuze gemaakt worden. Deze keuze wordt aan het lagere niveau overgelaten. Als er geen van de guards 'true' is, dan is dit een programmeerfout.

voorbeeld 1:
Het if-statement kan gebruikt worden om een keuze te maken uit een aantal opdrachten.

```
if A > 0 -> B:=2
[] A = 0 -> B:=1
[] A < 0 -> B:=0
fi
```

Er wordt dus een opdracht uitgevoerd. In dit voorbeeld kan er maar een guard 'true' zijn. Zoals hierboven is beschreven kunnen er bij het if-statement ook meer guards 'true' zijn waardoor er dan niet-determinisme op kan treden.

De tweede manier is het do....od paar

<statement> = do<guarded command set>od

Als geen van de guards 'true' is, wordt het statement overgeslagen (skip).
Zolang minstens een van de guards 'true' is, wordt een ervan gekozen en wordt diens statement list uitgevoerd. De volgorde waarin de guarded commands (waarvan de guards 'true' zijn) worden afgehandeld, moet worden gekozen zodat ook hier weer niet-determinisme optreedt.

voorbeeld 2:
Het do-statement kan gebruikt worden om de getallen Q1, Q2, Q3 en Q4 in opklimmende volgorde te ordenen in de variabelen q1, q2, q3 en q4. Dus na het do-statement geldt: q1 <= q2 <= q3 <= q4

```
q1,q2,q3,q4:=Q1,Q2,Q3,Q4;
do q1 > q2 -> q1,q2:=q2,q1
[] q2 > q3 -> q2,q3:=q3,q2
[] q3 > q4 -> q3,q4:=q4,q3
od
```

2.2 Waarom deze taal ?

De keuze van deze taal vraagt waarschijnlijk wel om een toelichting. Het gaat er in eerste instantie bij de beschrijving van het besturingsprogramma om dat gekeken wordt hoe de problemen aangepakt kunnen worden. Er hoeft in eerste instantie geen programma uit te komen dat direct op een computer kan draaien aangezien het schakelnetwerk ook nog niet aanwezig is. Het gaat om een eerste onderzoek naar de mogelijkheden.

2.2.1 nadelen

Als nadeel kan worden aangevoerd dat deze taal niet op een computer aanwezig is. Maar aangezien nog niet bekend is of het programma op een computer (en zoja op welke computer) zal moeten gaan draaien, heeft een bestaande computertaal dit nadeel ook behalve als net die computer gekozen wordt waarop die taal aanwezig is. De taal van Dijkstra is eigenlijk te beschouwen als een soort structuurdiagram. Als de computer en dus ook de taal bekend is kan het programma eenvoudig in die taal worden omgeschreven.
Als tweede nadeel kan nog genoemd worden dat de taal voor veel mensen onbekend zal zijn; dit nadeel is niet erg groot aangezien deze taal eenvoudig te lezen is.

2.2.2 voordelen

Tegenover de in de vorige subparagraaf genoemde nadelen staan een aantal voordelen.
Zo is het in deze taal mogelijk om op een zo hoog mogelijk niveau in eenduidige programma structuren te programmeren, ook kunnen soms bepaalde keuzes van datastructuren enz. vooreerst buiten beschouwing worden gelaten.

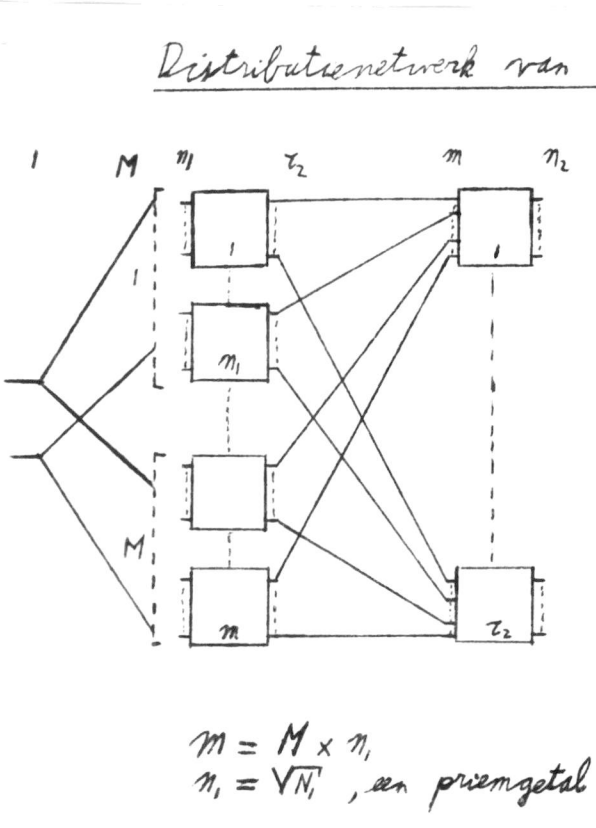
Het niet-determinisme voorkomt dat er onnodige keuzes gemaakt moeten worden.
En tenslotte zijn programma's die met deze taal zijn geschreven eenvoudig naar elke willekeurige computertaal om te schrijven.

3 DE BESTURING VAN HET DISTRIBUTIENETWERK; Het Richards netwerk

3.1 Inleiding

Een distributie netwerk kan in principe als een noncomposiete switch geimplementeerd worden. Er zijn dan N1*N2 kruispunten nodig met N1 het aantal stations en N2 het aantal abonnee's. Deze oplossing is voor grote N1 en N2 niet bruikbaar.

Het algemene Masson netwerk dat in Hoofdstuk 1 besproken is, kan met minder kruispunten (m*N2 + r2*N1*M) toe. Masson gebruikt een binomiale concentrator, deze is verre van ideaal.
Het meest effectieve distributienetwerk (de meest effectieve concentrator) dat tot nu toe bekend is, is het Richards netwerk [6]. De gegarandeerd blokkeringsvrije werking wordt slechts verkregen door het verleggen van bestaande verbindingen.
Hieronder volgt een schema van het netwerk:



figuur 5

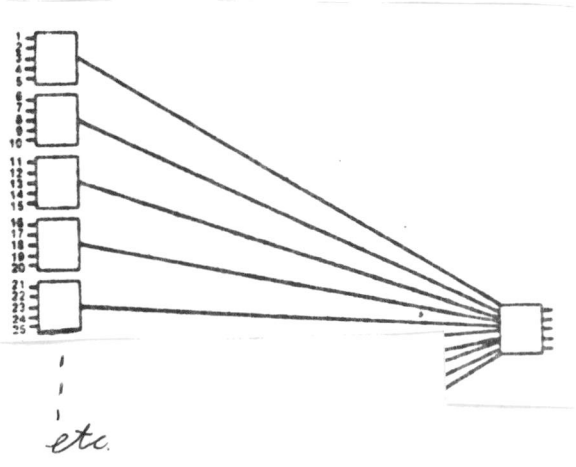
Er wordt op gewezen dat in dit verslag, in tegenstelling tot in het artikel van Richards en Hwang, bovenstaand netwerk beschouwd wordt als een drie traps netwerk, waarbij de ingangstrap een connectie trap is. Richards en Hwang noemen het netwerk een twee traps netwerk, hun ingangs trap is dus in dit verslag de midden trap.
Elk station komt M-voudig op de middentrap voor. De verdeling van de stations over de middenswitches is zodanig dat elk tweetal stations maar op een middenswitch samen aanwezig is.
Het aantal stations per ingangsswitch is n1, waarbij n1 de volgende waarde heeft:

$$n1 = SQRT(N1)$$

De stations komen M-voudig in een array met afmetingen n1 bij M*n1 te staan. De n1 getallen op een rij vormen de stations die op een bepaalde middenswitch voorkomen.

voorbeeld:
Dit voorbeeld toont een array en daaronder het netwerk met 25 stations (N1=25), 5 ingangen per ingangsswitch (n1=5), multipliciteit 2 (M=2) en 5 uitgangen (n2=5).

```
 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
- - - - - - - - - - - - -
 1 22 18 14 10
 6  2 23 19 15
11  7  3 24 20
16 12  8  4 25
21 17 13  9  5
```



figuur 6

In de eerste n1 rijen komt ieder station exact een keer voor.
Voor de formules zoals die gebruikt worden in het genoemde artikel gelden de volgende bereiken voor de betreffende variabelen:

kolomnummer c : $1 \leq c \leq n1$
rijnummer r : $1 \leq r \leq n1$
multipliciteitsnummer i : $1 \leq i \leq M$

Een gegeven station dat in kolom c en rij r in de eerste n1 rijen voorkomt, komt nog M-1 keer voor in kolom c en rij gegeven door :

$$1 + (i-1)*n1 + [r + c(i-1) - i] \bmod (n1) \qquad i=2,..,M \qquad (1)$$

Bovenstaande formule komt voor in het artikel van Richards en Hwang (zie ook bovenstaand voorbeeld en figuur 6). Deze formule is te vereenvoudigen door aanpassing van de volgende variabelen :

kolomnummer c' : $c' = c - 1,\ 0 \leq c' < n1$
rijnummer r' : $r' = r - 1,\ 0 \leq r' < n1$
multipliciteitsnummer i': $i' = i - 1,\ 0 \leq i' < M$

Dit geeft voor de rij nummers :

$$i'*n1 + [r' + c'*i'] \bmod (n1) \qquad i'=1,..,M-1 \qquad (1')$$

Door voor n1 een priemgetal te nemen komt elk tweetal stations maar op een rij samen voor en dus ook maar op een middenswitch (zie ook bovenstaand voorbeeld).
De maximale waarde voor n2 (het aantal uitgangen op een eindswitch) is niet in z'n algemeenheid te geven voor deze verdeling van de stations over de middentrap. Wel kan er een onder- en bovengrens worden gevonden (zie artikel van Richards en Hwang). De ondergrens is die waarde waarbij het netwerk zeker nog niet blokkeert. En de bovengrens is die waarde waarboven het netwerk zeker blokkerende toestanden heeft. Een "veilige" bovengrens is dus tevens een ondergrens.
Een ondergrens voor n2 is (uit het artikel van Richards en Hwang):

$$M(M+1)-1 \qquad (2)$$

Voor M=2 is de bovengrens voor n2 5 en bij M=3 is deze bovengrens 13.
Voor waarden van $M > 3$ is een bovengrens gegeven door:

$$\frac{2}{3}M^3 - 2M^2 + \frac{16}{3}M - 3 \qquad (3)$$

Het aantal kruispunten in het Richards netwerk is $N1*N2[M(1/n1 + 1/n2)]$

Er moet dus een minimum worden gevonden voor :$M(1/n1 + 1/n2)$ (4)

Om nu het aantal kruispunten te minimaliseren worden de waarden van M en de bovengrens van n2 (uit (3)) ingevuld in (4). Hierbij wordt in genoemd artikel aangenomen dat de bovengrens (uit (3)) "veilig" is. Hieronder volgen enige waarden:

aantal stations	beste waarde voor M	aantal uitgangen op eindswitch
N1<=34	2	5
35<=N1<=116	3	13
117<=N1<=396	4	29
397<=N1<=1247	5	57

Het is mogelijk om bij grote netwerken een verdere vermindering van het aantal kruispunten te krijgen door recursie, dat wil zeggen individuele switches vervangen door het Richards netwerk.

3.2 De besturing van het Richards netwerk

Het is voldoende om het Richards netwerk te bekijken met slechts een eindswitch, elke eindswitch is afzonderlijk verbonden met de middenswitches zodat nooit een bepaalde eindswitch een andere eindswitch kan blokkeren.
Als een abonnee een bepaald station vraagt kunnen er zich twee situaties voordoen : namelijk, of het station is al op de eindswitch aanwezig en dan hoeft de abonnee alleen in de eindswitch doorgeschakeld te worden, of het station is nog niet op de eindswitch aanwezig en dan moet er een vrij pad gezocht worden van station naar abonnee. Als er geen vrij pad te vinden is dan zullen er eerst verleggingen nodig zijn alvorens de nieuwe aanvraag afgehandeld kan worden.

Voor grotere netwerken met meer dan drie trappen gaat het zoeken van een vrij pad recursief.

3.3 Verdeling van de programma's

In Hoofdstuk 4 zullen de drie programma's besproken worden. In het eerste programma wordt er van uitgegaan dat er geen verleggingen nodig zijn, dus elke oproep is te leggen. Vervolgens wordt in het tweede programma het verleggen uitgewerkt waarbij nog niet gepoogd wordt dit verleggen zo slim/snel mogelijk te doen, er wordt alleen verlangd dat het verleggen eindigt.
Tenslotte wordt dan in het derde programma het verleggen zo slim/snel mogelijk gedaan.

4 BESCHRIJVING VAN DE DRIE PROGRAMMA'S

In de volgende drie paragrafen worden de drie programma's besproken.

4.1 Het eerste programma

Er wordt bij dit programma van uitgegaan dat er geen verleggen nodig is. Het programma bestaat uit een oneindige lus waarin gekeken wordt of er een oproep (call_request) danwel een vrijgave (call_release) binnenkomt. Als er een oproep aanwezig is, is de eerste vraag of het gevraagde station al aanwezig is op de eindswitch, zoja dan hoeft alleen de betreffende uitgang doorverbonden te worden.
Als het gevraagde station niet aanwezig is moet een vrije weg van het station naar de eindswitch gezocht worden. Hierna moet dan een van de mogelijkheden worden gekozen.
De situatie "rearranging" doet zich hier dus niet voor. In het tweede programma (=uitbreiding eerste programma, zie paragraaf 4.2) komt dit aan de orde.
Als er een vrijgave aanwezig is, moet de output vrijgemaakt worden. Vervolgens moet worden gekeken of de abonnee de enige gebruiker was, zoja dan moet het station van de eindswitch verwijderd worden.
Hieronder volgt het programma, de teksten tussen " " worden daarna besproken.

```
begin
    glovar call_request,call_release;
    glovar stat_req,outp_req,outp_rel;
    glocon M;
    privar counter,x,pos;
    privar station,multipl;
    privar reach_paths;
    privar midswitch,secsw_input,endsw_input;
    privar always;
    always vir bool:=true;
    call_request vir bool,call_release vir bool:=false,false;
    "initialisation";
    do always ->
        "look for call_request and/or call_release";
        do call_request ->
            "inlet available ?";
            if "inlet available" ->
                "connect outlet"
            [] "inlet not available" ->
                "which paths reachable";
                if reach_paths > 0  ->
                    "take one"
                [] reach_paths = 0  ->
                    "rearranging"
                fi;
                 "connect chosen path"
            fi;
            output(outp_req):=stat_req;
```

```
            call_request:=false
        [] call_release ->
            "make output free";
            if "output only station-user" ->
                "remove station from endswitch"
            [] "output not only station-user" ->
                skip
            fi;
            call_release:=false
        od
    od
end
```

"initialisation"

Opmerking: middenswitch p is verbonden met ingang p van de eindswitch

In de volgende array-variabelen wordt de toestand van het netwerk bijgehouden:

- stat_avail : variabel
  functie: het bijhouden van de op de eindswitch aanwezige stations
  input : type station
  output : type multiplicity of nil
  geinitialiseerd op nil
- inp_of_endsw : variabel
  functie: het bijhouden welke middenswitch vrij zijn
  input : type middenswitch
  output : element van {free,busy}
  geinitialiseerd op free
- pos_paths : variabel
  functie: opslaan van de vrije mogelijkheden
  input : type integer (0,..3)
  output : type multiplicity of nil
  geinitialiseerd op nil
- user_counter : variabel
  functie: tellen van het aantal gebruikers van een bepaalde ingang van de eindswitch
  input : type middleswitch
  output : type integer
  geinitialiseerd op 0
- output : variabel
  functie: aangeven met welk station betreffende abonnee (outp_rel) verbonden is
  input : type output
  output : type station of nil
  geinitialiseerd op nil

Tijdens de initialisatie moeten de gegevens van de verdeling van de stations over de secondswitches opgeslagen worden in de volgende

'variabelen':

- stat_on_secsw: constant
  functie: aangeven op welke middenswitches het station aanwezig is
  input : type station,multiplicity
  output : type middleswitch
  geinitialiseerd op verdeling stations
- inp_of_secsw : constant
  functie: aangeven op welke ingang van de middenswitch het station aanwezig is
  input : type station,multiplicity
  output : type inp_secsw
  geinitialiseerd op verdeling stations

"look for call_request and/or call_release"

Hoe de eindswitch aangeeft wanneer er een call_request danwel call_release binnenkomt wordt vooreerst buiten beschouwing gelaten. Een call_request heeft tot gevolg dat de volgende variabelen een (nieuwe) waarde krijgen:

- stat_req : type station
- outp_req : type output

En evenzo heeft een call_release tot gevolg dat de volgende variabele een (nieuwe) waarde krijgt:

- outp_rel : type output

"inlet available ?"

Het al dan niet aanwezig zijn van het gevraagde station op de eindswitch wordt gehaald uit de variabele stat_avail indien : deze variabele de waarde -inf heeft, betekent dit dat het gevraagde station niet aanwezig is.

(Een overweging is om het aanwezig zijn en zoja op welke input (van de eindswitch) in twee gedeelten te splitsen, en niet zoals nu in een variabele.)

In de andere gevallen geeft deze variabele de multipliciteit van het station aan.

multipl:=stat_avail(stat_req)

"inlet available"

Als de variabele multiplicity positief (>=0) is, is het gevraagde station aanwezig.

multipl >= 0

"inlet not available"

De waarde -inf duidt op het niet aanwezig zijn van het station.

```
multipl = -inf
```

"which paths reachable"

Hier moet worden onderzocht welke paden er vrij zijn, dus welke er niet voor andere calls worden gebruikt.
Maximaal zijn er M mogelijke paden. Deze mogelijkheden worden allemaal onderzocht, en via de variabele counter in een array gezet.
De variabele stat_on_secsw geeft aan op welke middenswitches een station aanwezig is.
Als de betreffende middenswitch en dus input van eindswitch niet vrij is hoeft er niets gedaan te worden (skip, zie paragraaf 2.1).

```
counter:=0;
reach_paths:=0;
do counter < M  ->
    pos:=stat_on_secsw(stat_req;counter);
    if inp_of_endsw(pos)=free ->
        "put multiplicity in array"
    [] inp_of_endsw(pos)=busy ->
        skip
    fi;
    counter:=counter + 1
od
```

"put multiplicity in array"

Als er een vrije kandidaat gevonden is, wordt deze in een array (pos_paths) opgeslagen.

```
pos_paths(reach_paths):=counter;
reach_paths:=reach_paths + 1
```

"connect outlet"

Hiermee wordt de uitgang verbonden met het gevraagde station als dat station al op de eindswitch aanwezig is. De opdracht connectend verbindt de oproeper met het gewenste station.

- connectend : functie : in eindswitch verbinden van ingang en uitgang (=oproeper)
  input : type middleswitch,output
  output : geen

De variabele user_counter houdt bij hoeveel abonnees van de betreffende input en dus van het betreffende station gebruik maken. Dit om te weten wanneer een bepaalde input niet meer gebruikt wordt en dus van de eindswitch verwijderd kan worden.

```
endsw_input:=stat_on_secsw(stat_req;multipl);
connectend(endsw_input;outp_req);
user_counter(endsw_input):=user_counter(endsw_input) + 1
```

"take one"

Hier wordt een van de mogelijkheden gekozen. De variabele x heeft hier een willekeurige waarde in het interval [0,reach_paths>.

```
x:=choice([0,reach_paths>);
multipl:=pos_paths(x)
```

"connect chosen path"

Er moet nu in de secondswitch en in de eindswitch een verbinding worden gelegd. De functie connectend is bij "connect outlet" al besproken.
Voor de verbinding in de secondswitch:

- connectmid : functie : in middenswitch verbinden van station en uitgang (=ingang eindswitch)
  input : type middleswitch,inp_secsw
  output : geen

Verder moet de teller (van het aantal gebruikers van die ingang) met 1 worden verhoogd. Tenslotte moet de ingang van de eindswitch bezet geboekt worden.

```
midswitch:=stat_on_secsw(stat_req;multipl);
secsw_input:=inp_of_secsw(stat_req;multipl);
connectmid(midswitch;secsw_input);
endsw_input:=midswitch;
connectend(endsw_input;outp_req);
user_counter(endsw_input):=user_counter(endsw_input) + 1;
stat_avail(stat_req):=multipl;
inp_of_endsw(endsw_input):=busy
```

"make output free"

Als er een call_release binnenkomt, moet in ieder geval de output vrij gemaakt worden.

```
- disconnectend : functie : in  eindswitch  verbreken van ingang
                            en uitgang (= outp_release)
                  input   : type middleswitch,output
                  output  : geen
```

Verder moet de teller die aangeeft hoeveel abonnees het betreffende station gebruiken met een verlaagd worden.

```
station:=output(outp_rel);
output(outp_rel):= -inf;
multipl:=stat_avail(station);
endsw_input:=stat_on_secsw(station;multipl);
disconnectend(endsw_input;outp_rel);
user_counter(endsw_input):=user_counter(endsw_input) - 1
```

"output only station_user"

Als de teller na het verlagen 0 is, was de call (van de call_release) de enige die dat station gebruikte.

```
user_counter(endsw_input) = 0
```

"remove station from endswitch"

Als er geen gebruiker meer is voor het betreffende station dan moet dit station van de eindswitch verwijderd worden.

```
disconnectmid : functie : in middenswitch verbreken  van station
                          en uitgang (=ingang eindswitch)
                input   : type middleswitch,inp_secsw
                output  : geen
```

De negatieve waarde van de variabele stat_avail geeft aan dat het betreffende station niet meer op de eindswitch aanwezig is. Tenslotte wordt deze input van de eindswitch vrij geboekt.

```
multipl:=stat_avail(station);
secsw_input:=inp_of_secsw(station;multipl);
midswitch:=endsw_input;
disconnectmid(midswitch;secsw_input);
stat_avail(station):= -inf;
inp_of_endsw(endsw_input):=free
```

"output not only station_user"

Het groter dan 0 zijn van de variabele user_counter geeft aan dat er meer gebruikers van dat station waren.

```
user_counter(endsw_input) > 0
```

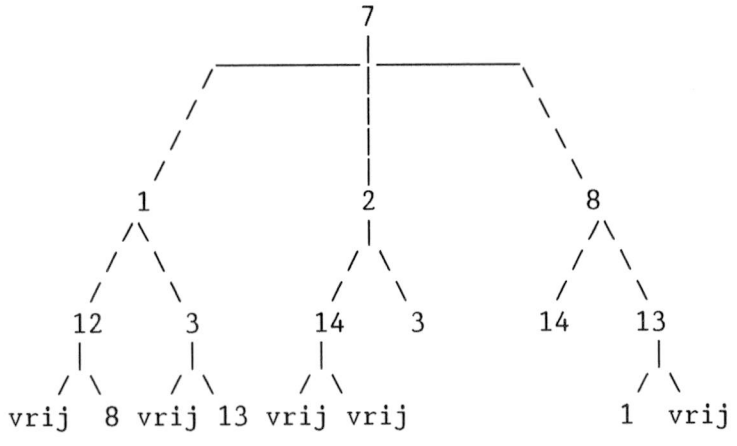
Hiermee is het eerste programma volledig besproken. In paragraaf 4.2 wordt het tweede programma besproken. De gedeelten die in het tweede programma gelijk zijn aan het eerste programma worden niet herhaald, er wordt dan naar deze paragraaf verwezen.

4.2 Het tweede programma

In dit tweede programma wordt het verleggen toegevoegd, waarbij nog niet slim/snel verleggen in beschouwing wordt genomen. Het gedeelte "rearranging" (zie het eerste programma, paragraaf 4.1) wordt nu dus ingevuld.

Als er geen ingang (van de eindswitch) meer vrij is voor het betreffende station moet door middel van verleggingen een vrije ingang verkregen worden.
De manier van verleggen lijkt veel op de structuur zoals die, heel in het kort, beschreven staat in het artikel van Richards [6]. Daar wordt die structuur een tree genoemd, deze benaming is onjuist. Wiskundig gezien is een tree een graph zonder cycles, terwijl er in de tree zoals Richards die gebruikt wel cycles voor kunnen komen. Er is voor gekozen om toch hier de benaming tree te gebruiken, misschien zou stamboom een betere benaming zijn. Zie voorbeeld 1 (voorbeeld uit het artikel van Richards).

voorbeeld 1:
Station 7 moet gelegd worden, maar de 3 plaatsen (M=3) waarop dat zou kunnen, worden geblokkeerd door resp. station 1,2 en 8. Die op hun beurt weer worden geblokkeerd door resp. {12,3}, {14,3} en {14,13}. Een niveau lager komen er plaatsen vrij en kan het eigenlijke verleggen plaats vinden.



(einde voorbeeld 1)

Het verleggen bestaat nu uit het vinden van een vrije mogelijkheid om een station op een andere manier te leggen. Dit kan op verschillende manieren, zo kan b.v. gelijkmatig over de tree steeds dieper worden gezocht. Of, en dat wordt hier toegepast, kan bij elke vertakking van de tree een keuze worden gemaakt (willekeurig).
Als er een cycle ontstaat, een station dat twee keer voorkomt, wordt er een stap terug gedaan.

Zo wordt er een pad gevonden van het gevraagde station via een aantal blokkerende stations naar uiteindelijk een mogelijkheid om een station direct te verleggen. Daarna wordt het pad in omgekeerde richting doorlopen tot uiteindelijk het gevraagde station (stat_req) gelegd kan worden.

Stelling 1:

Bij het Richards-netwerk kan zich geen cycle ter lengte 2 voordoen, omdat elk tweetal stations maar op hoogstens 1 middenswitch samen aanwezig is. De minimale cycle-lengte is hier dan ook 3.

Stelling 2:

In een herarrangeerbaar algemeen Masson netwerk hoeft voor een verbindingsaanvrage een bestaande verbinding nooit meer dan eens te worden verlegd. Dit is een gevolg van : - verlegging-stappen sequentieel - een station wordt slechts op 1 middenswitch doorgeschakeld - verhinderen van cycles

Het bewijs van deze stellingen wordt gegeven in Bijlage 1.

De variabele station heeft de waarde van het station (input) dat gelegd/verlegd moet worden. Verder geeft de variabele stat_itself de multipliciteit aan waar de betreffende input (variabele station) zelf op aanwezig is en dus waar vandaan hij weg moet om plaats te maken voor de voorgaande in de rij (variabele patharray).
De boolean variabele blocked heeft de waarde 'true' zolang er nog geen vrije mogelijkheid gevonden is. Zolang de conditie blocked true is, wordt de do-loop doorlopen. Indien de boolean-variabele cycle false is, wordt een volgend element voor de rij gekozen ("choose one and put in patharray"). Anders wordt een stap terug gezet ("one step back") en wordt er een andere keuze gemaakt ("other choice").
Vervolgens wordt er gekeken of er een cycle optreedt ("check on cycle").

En tenslotte wordt met behulp van een if-statement, als er geen cycle is opgetreden, weer gekeken door welke stations een mogelijkheid wordt geblokkeerd ("blocked by which stations").
Aangezien de eerste keer dat de do-loop doorlopen wordt er geen cycle kan zijn opgetreden, is deze volgorde geoorloofd. Het voordeel van deze methode is dat alleen gekeken wordt welke stations een mogelijkheid blokkeren als er geen cycle is opgetreden, zodat geen onnodig en dubbel werk wordt gedaan.
Wanneer de conditie blocked false wordt, is er minstens 1 vrije mogelijkheid gevonden en kan het eigenlijke verleggen beginnen ("rear"). Er wordt nog een extra zekerheid ingebouwd voor het geval aan het netwerk meer dan de toegestane hoeveelheid oproepen wordt aangeboden (zie "one step back").

Hieronder volgt het gedeelte "rearranging", de teksten tussen " " worden daarna besproken.

```
"rearranging"

begin
   glocon M,stat_req;
   glovar always,multipl,midswitch,secsw_input,endsw_input;
   glovar station,counter,x;
   privar blocked,cycle,end_of_path,secsw,candi;
   privar stat_itself,freepos,new_endsw_input,y;
   pricon cycle_min,n2;
   "init rearranging";
   station:=stat_req;
   stat_itself:=nil;
   blocked:=true;
   freepos:=0;
   cycle:=false;
   "put stat_req in patharray";
   "blocked by which stations";
   do blocked ->
               if not cycle -> "choose one and put in patharray"
               [] cycle     -> "one step back";
                               "other choice"
               fi;
               "check on cycle";
               if not cycle -> "blocked by which stations"
               [] cycle     -> skip
               fi
   od;
   "rear"
end
```

"init rearranging"

Voor het bijhouden van de toestand van de rearranging zijn nog de volgende variabelen nodig:

```
patharray            :variabel
                      functie: opslaan pad van blokkerende stations
                               met multipliciteiten.
                      input  : type integer
                      output : type station, multiplicity of nil
                      geinitialiseerd op nil
```

Een voorbeeld van de werking van patharray:

```
        plaats 0 in patharray    : stat_req
               1                 : multipliciteit(stat_req)
               2                 : blokkerend station 1
               3                 : multipl.(blokk.station 1)
               4                 : blokkerend station 2
               5                 : multipl.(blokk.station 2)
               6                 : blokkerend station 3
               7                 : etc.
```

```
inv_stat_avail      :variabel
                     functie: inverse van stat_avail waarbij extra de
                              betreffende middenswitch nodig is (zie
                              eerste programma, paragraaf 4.1).
                     input  : type middleswitch,multiplicity
                     output : type station of nil

freearray            :variabel
                     functie: opslaan multipliciteiten van de vrije
                              mogelijkheden.
                     input  : type integer
                     output : type multiplicity of nil
                     geinitialiseerd op nil

blockarray           :variabel
                     functie: tijdelijk opslaan van blokkerende
                              stations
                     input  : type multiplicity
                     output : type station of nil
                     geinitialiseerd op nil
```

"put stat_req in patharray"

In de variabele patharray komt aan het begin het gevraagde station (stat_req) te staan. Met de teller end_of_path wordt de laatste plaats in de rij aangegeven (zie b.v. ook "choose one and put in patharray").

```
end_of_path:=0;
```

```
patharray(end_of_path):=stat_req
```

"blocked by which stations"

Met het gedeelte "blocked by which stations" wordt bepaald door welke inputs het gevraagde station (dat in variabele station staat) geblokkeerd wordt. Deze blokkerende stations worden tijdelijk in de variabele blockarray gezet ("put blocking in blockarray").
De teller counter doorloopt het waardenbereik 0 t/m M-1. Met de statements

```
secsw := stat_on_secsw(station;counter);
candi := inv_stat_avail(secsw;counter)
```

wordt het station bepaald dat op de betreffende input (van de eindswitch) aanwezig is en wordt dit station in de variabele candi gezet.
Als candi gelijk is aan de variabele station dan is het station op deze multipliciteitwaarde zelf aanwezig. En moet dus verlegd worden om plaats te maken voor de voorgaande in de rij. Met behulp van de variabele triedmultipl wordt per station bijgehouden welke multipliciteiten al geprobeerd zijn (zie ook "one step back").
Als candi gelijk is aan nil dan is er een vrije mogelijkheid gevonden. Deze moet dan in een array/rij geplaatst worden ("put possibility in freearray").
En tenslotte als candi ongelijk is aan station en ongelijk aan nil dan moet deze in de variabele blockarray worden gezet ("put blocking in blockarray").

```
counter:=0;
do counter < M  -> secsw:=stat_on_secsw(station;counter);
                   candi:=inv_stat_avail(secsw);
                   if candi = station ->
                        stat_itself:=counter;
                        triedmultipl(station;0):=stat_itself
                   [] candi = nil     ->
                        "put possibility in freearray";
                        blocked:=false
                   [] candi <> station  and  candi <> nil ->
                        "put blocking in blockarray"
                   fi;
                   counter:=counter + 1
od
```

"put possibility in freearray"

```
freearray(freepos):=counter;
freepos:=freepos + 1
```

In de variabele counter staat de multipliciteit van de vrije mogelijkheid.

"put blocking in blockarray"

```
blockarray(counter):=candi
```

Op de multipliciteitswaarde, die in de variabele counter is opgeslagen, wordt het blokkerende station bewaard.

"check on cycle"

Om te voorkomen dat het verleggen in een cycle terecht komt, moet elke keer als er een nieuw element aan de rij (patharray(end_of_path)) wordt toegevoegd gekeken worden of dat station al eerder in de rij voorkomt.
Omdat de minimale cyclelengte 3 is bij het Richards-netwerk en omdat van de stations ook de multipliciteit moet worden opgeslagen hoeft de do-loop alleen doorlopen te worden als de variabele end_of_path groter dan 6 (2 * cycle-lengte) is en er nog geen cycle geconstateerd is. De teller counter doorloopt de tweevouden vanaf 0 tot en met (end_of_path - cycle_min - 2).
De eerste waarde in patharray (=stat_req) hoeft niet onderzocht te worden omdat dat station zeker nog niet voorkomt.
Als er een station wordt gevonden dat al eerder in de tree voorkomt wordt de boolean-variabele cycle op de waarde true gezet.

```
cycle:=false;
counter:=0;
cycle_min:=6; { minimale cycle-lengte 3 voor Richards }
do counter < (end_of_path - cycle_min)  and  not cycle ->
          counter:=counter + 2;
          if patharray(end_of_path) = patharray(counter)  ->
                              cycle:=true
          [] patharray(end_of_path) <> patharray(counter) -> skip
          fi
od
```

"choose one and put in patharray"

Als er geen cycle is geconstateerd moet een keuze gemaakt worden uit een van de M-1 mogelijkheden (M mogelijkheden maar op 1 daarvan zit het station zelf). Dit gebeurt met de functie choice. De variabele stat_itself geeft de multipliciteitswaarde aan waar het blokkerende station zelf op zit en dus waarvan het moet verdwijnen.
Achtereenvolgens worden dan de multipliciteitswaarde en het blokkerende station in de variabele patharray gezet.
De variabele station krijgt nu weer de waarde van het laatste blokkerende station in de rij.

```
x:=choice([0,M-1] / stat_itself);
end_of_path:=end_of_path + 1;
patharray(end_of_path):=x;
end_of_path:=end_of_path + 1;
patharray(end_of_path):=blockarray(x);
station:=patharray(end_of_path)
```

"one step back"

Na het constateren van een cycle moet een andere keuze gemaakt worden. Er moet worden bijgehouden welke mogelijkheden al geprobeerd zijn en dus niet nog een keer gekozen mogen worden.
Hiervoor zijn de volgende variabelen nodig:

```
multicount      :variabel
                 functie: tellen aantal geprobeerde multipliciteiten
                 input  : type station
                 output : type integer
                 geinitialiseerd op 0

triedmultipl    :variabel
                 functie: opslaan multipliciteitswaarden die al
                          geprobeerd zijn
                 input  : type station,integer
                 output : type multiplicity of nil
                 geinitialiseerd op nil
```

Eerst wordt de boolean-variabele cycle op false gezet. Vervolgens wordt de laatste waarde uit het pad verwijderd, immers die veroorzaakte de cycle. Daarna moet de multipliciteitswaarde worden opgeslagen in de variabele triedmultipl. Om dat te bereiken is het station nodig dat daarvoor in het pad staat. Als de multipliciteitswaarde is opgeslagen wordt die uit het pad verwijderd. Aan het eind van het pad (end_of_path) staat weer het laatste blokkerende station.
Vervolgens zijn er twee mogelijkheden, nl:

1. Er zijn nog niet-geprobeerde multipliciteiten over.
2. Deze mislukte multipliciteit was de enige nog niet geprobeerde.

Bij mogelijkheid 1 hoeft er verder niets gedaan te worden, ook "blocked by which stations" is hier niet nodig omdat deze blokkerende stations nog in de variabele blockarray staan.
Echter bij mogelijkheid 2 moet nogmaals een stap terug worden gezet, dit moet herhaald worden totdat er een mogelijkheid is gevonden om een nieuwe tak in de tree in te gaan. Ook hier moet weer de laatste waarde uit het pad worden verwijderd, de multipliciteitswaarde worden opgeslagen en de variabele end_of_path weer naar het laatste blokkerende station laten wijzen.
Als er een mogelijkheid wordt gevonden (multicount(station) < M - 1) wordt de do-loop verlaten en wordt er gekeken welke stations deze mogelijkheid blokkeren.

Indien het netwerk met teveel oproepen wordt belast, is het mogelijk dat er geen pad te vinden is met aan het eind een vrije mogelijkheid. In dat geval moet het programma een mededeling geven en stoppen (STOP PROGRAM). Er is geen zekerheid dat het netwerk niet kan blokkeren.

```
cycle:=false;
patharray(end_of_path):=nil;
end_of_path:=end_of_path - 1;
station:=patharray(end_of_path - 1);
multicount(station):=multicount(station) + 1;
triedmultipl(station;multicount(station)):=patharray(end_of_path);
patharray(end_of_path):=nil;
end_of_path:=end_of_path - 1;
if multicount(station) < M-1 -> skip
[] multicount(station) = M-1 ->
    do multicount(station) = M-1 ->
           multicount(station):=0;
           patharray(end_of_path):=nil;
           end_of_path:=end_of_path - 1;
           station:=patharray(end_of_path - 1);
           multicount(station):=multicount(station) + 1;
           triedmultipl(station;multicount(station)):=
                              patharray(end_of_path);
           patharray(end_of_path):=nil;
           end_of_path:=end_of_path - 1
           if end_of_path = 0 -> always:=false;
                                 print"capaciteit van netwerk
                                       overschreden, programma
                                       gestopt!";
                                 STOP PROGRAM
           [] end_of_path > 0 -> skip
           fi
    od;
    "blocked by which stations"
fi
```

"other choice"

Het gedeelte "other choice" verschilt van "choose one and put in patharray" doordat er nu met choice een keuze gemaakt moet worden waarbij de multipliciteitswaarden die al geprobeerd zijn niet nogmaals mogen worden gekozen. Dit wordt gedaan door gebruik te maken van de conditie
0 <= y <= multicount(station). Waarbij op y=0 de multipliciteitswaarde van het station zelf aanwezig is.

Zie voor verdere uitleg "choose one and put in patharray".

```
x:=choice([0,M-1] / triedmultipl(station;y)| 0 <= y <=
                                              multicount(station));
end_of_path:=end_of_path + 1;
patharray(end_of_path):=x;
end_of_path:=end_of_path + 1;
patharray(end_of_path):=blockarray(x);
station:=patharray(end_of_path)
```

"rear"

In het gedeelte "rear" wordt het eigenlijke verleggen van al bestaande verbindingen, ten behoeve van de nieuwe aanvraag, gedaan. De nieuwe aanvraag zelf (stat_req,outp_req) wordt niet in het gedeelte "rearranging" gelegd, maar wordt net als een aanvraag die direct te leggen is in het gedeelte "connect chosen path" gelegd (zie eerste programma, paragraaf 4.1).
Eerst moet er een keuze gemaakt worden welke multipliciteit genomen wordt. Deze zijn opgeslagen in de variabele freearray.
Vervolgens wordt het station van het eind van het pad gehaald en kan met het verleggen worden begonnen (grote do-loop). Totdat het begin van het pad wordt bereikt (= toe aan leggen van nieuwe oproep) wordt deze do-loop doorlopen.
In deze grote do-loop wordt eerst het betreffende station in de middenswitch dubbel gelegd. De kleine do-loop wordt vervolgens gebruikt om de abonnee's in de eindswitch te verleggen.Hiervoor is een extra variabele nodig:

```
known          :variabel
                functie: er voor zorgen dat vrije abonnee's maar 1
                         keer worden bekeken en al verlegde abonnee's
                         verder over worden overgeslagen
                input  : integer
                output : element van {false,true}
                geinitialiseerd op false
```

Verder wordt dan in deze grote do-loop de oude verbinding in de middenswitch verwijderd en wordt het volgende te verleggen station uit het pad gehaald.

```
x:=choice([0,freepos-1]);
multipl:=freearray(x);
station:=patharray(end_of_path);
multicount(station):=0;
end_of_path:=end_of_path - 1;
do end_of_path >= 1 ->
   midswitch:=stat_on_secsw(station;multipl);
   secsw_input:=inp_of_secsw(station;multipl);
   connectmid(midswitch;secsw_input);
   counter:=stat_avail(station);
   endsw_input:=stat_on_secsw(station;counter);
```

```
        new_endsw_input:=midswitch;
        counter:=0;
        do counter < n2 ->
           if known(counter)      -> skip
           [] not known(counter) ->
              if output(counter) <> station  and output(counter) <> nil ->
                                                                  skip
              [] output(counter) = nil      -> known(counter):=true
              [] output(counter) = station ->
                                    known(counter):=true;
                                    connectend(new_endsw_input;counter);
                                    disconnectend(endsw_input;counter)
              fi
           fi;
           counter:=counter + 1
        od;
        midswitch:=endsw_input;
        secsw_input:=inp_of_secsw(station;triedmultipl(station;0));
        disconnectmid(midswitch;secsw_input);
        user_counter(new_endsw_input):=user_counter(endsw_input);
        user_counter(endsw_input):=0;
        stat_avail(station):=multipl;
        inp_of_endsw(endsw_input):=free;
        inp_of_endsw(new_endsw_input):=busy;
        multipl:=patharray(end_of_path);
        end_of_path:=end_of_path - 1;
        station:=patharray(end_of_path);
        multicount(station):=0;
        end_of_path:=end_of_path - 1
     od;
```

Hiermee is het tweede programma volledig. In de volgende paragraaf wordt gekeken hoe het leggen en verleggen zo slim/snel mogelijk kan geschieden.

4.3 Het derde programma

In dit programma moet het leggen van nieuwe aanvragen en het verleggen van bestaande verbindingen zo slim en snel mogelijk gebeuren.Er zal gekozen moeten worden welke manier van leggen en verleggen van calls gebruikt gaat worden.
Aangezien er geen simulatieresultaten voorhanden zijn, zal deze keuze meer op vermoedens aan de hand van uitgewerkte voorbeelden genomen worden dan dat die keuze echt onderbouwd kan worden.

Er zijn verschillende manieren om het aantal keren dat er verlegd moet worden te beperken:

1. Als de meest gevraagde stations bekend zijn, kan hier enig voordeel uit worden verkregen. Namelijk door deze stations op verschillende middenswitches beschikbaar te stellen. Dit voordeel

is echter heel beperkt, zeker omdat op dit moment nog niets over de stations bekend is en doordat met de komst van allerlei videotoepassingen deze meest gevraagde stations heel sterk kunnen wisselen.

2. Ook kan geprobeerd worden de oproepen door het netwerk zo scheef mogelijk te verdelen (="aanstampen"). Dit kan nog weer op twee manieren, namelijk kan geprobeerd worden dit aanstampen per middenswitch te doen of men kan dit per subarray (multipliciteit) proberen te bereiken. Aangezien bij de beschrijving van het Richards-netwerk uit wordt gegaan van 1 eindswitch heeft kijken naar de meest belaste middenswitch hier geen zin.
   Bij het aanstampen per subarray blijven andere multipliciteiten licht belast. Zolang er 1 subarray helemaal leeg is, zijn er geen verleggingen nodig.

3. Tenslotte kan worden bijgehouden hoeveel middenswitches voor elk station bezet zijn en het zo uniform mogelijk leggen van de oproepen ten aanzien van de switches voor elk station.
   Dit houdt in dat er naar gestreefd wordt dat voor elk station evenveel multipliciteiten vrij blijven. Hierdoor blijft elk station zolang mogelijk beschikbaar. Als een switch de enige overgebleven mogelijkheid is voor een bepaald ongebruikt station, dan wordt deze mogelijkheid voor het leggen van een ander station vermeden behalve als er geen andere mogelijkheid meer is. Zolang er tenminste 1 switch vrij is voor ieder niet-gebruikt station zullen er geen verleggingen nodig zijn.
   Het vermoeden bestaat dat het verleggen, als dat noodzakelijk wordt, beperkt zal blijven tot een gering aantal omdat per station een zo uniform mogelijke hoeveelheid multipliciteiten beschikbaar blijft.

Hieronder volgen nog een aantal overwegingen ten aanzien van verleggingen:

- Zal aanstampen extra lange verleggingen geven ?

- In plaats van 1 tak in de tree te kiezen, uniform over tree steeds dieper gaan zoeken, sneller resultaat ?

- Richards en Hwang hebben tijdens simulaties voor het aantal te verleggen verbindingen hooguit M gevonden. En dus, omdat elke verbinding maar ten hoogste 1 keer hoeft te worden verlegd, zijn er ten hoogste M verleggingen om een nieuwe aanvraag te leggen.

het leggen van nieuwe aanvragen

Zoals in het overzicht is aangegeven zijn er twee bruikbare methoden om nieuwe aanvragen te leggen:

1. aanstampen

2. uniform over de multipliciteiten leggen

Met voorbeelden is gekeken welke van de twee methoden het beste is. Het blijkt dat aanstampen en uniform leggen veel overeenkomst vertonen, waarbij alleen soms aanstampen en heel soms uniform leggen wat gunstiger is.

Vermoeden 1:

Door de strategie aanstampen kan het aantal keren dat verlegd moet worden beperkt blijven.

In Bijlage 2 staan een aantal voorbeelden waar o.a. dit vermoeden op gebaseerd is.

De voorkeur wordt nu gegeven om als "strategie" aanstampen te gebruiken, dit heeft namelijk ook voordelen bij de implementatie. Voor de strategie uniform leggen zijn veel meer variabelen etc. nodig om alle gegevens bij te houden.

het verleggen

Om bij het verleggen ook zo verstandig mogelijk te werken, worden hier ook wijzigingen aangebracht. In het tweede programma werd in elke vertakking van de tree een willekeurige keuze gemaakt. Dat was de eenvoudigste maar zeker niet de beste keuze.
Een andere methode zou zijn, zoals bij het tweede programma al werd besproken, uniform over de tree steeds dieper te gaan zoeken. Dit heeft echter het nadeel dat veel administratief werk gedaan moet worden, namelijk moet elke tak van de tree worden bijgehouden.
Er is nog een derde methode en die maakt wel op elke vertakking van de tree een keuze maar nu een veel betere keuze:

Vermoeden 2:

Door bij elke vertakking van de tree, als er nog geen vrije mogelijkheid is gevonden, de zwaarst belaste (=drukst bezette) multipliciteit te kiezen, is de kans het grootst dat op een niveau lager een vrije mogelijkheid wordt gevonden.

In Bijlage 1 wordt een toelichting op dit vermoeden gegeven.

Al zal bij deze keuze misschien niet in alle gevallen het kortste pad worden gevonden, het levert snel en eenvoudig een oplossing voor het verleggen.

de programma beschrijving

Hieronder volgt de bespreking van het volledige programma. Het hoofdprogramma is voor een gedeelte gelijk aan het eerste programma. Dit programma heeft dezelfde structuur als het eerste programma.
De volgende gedeelten zijn hetzelfde als in het eerste programma (voor toelichting zie paragraaf 4.1):

- "look for call_request and/or call_release"
- "inlet available ?"
- "inlet available"
- "inlet not available"
- "which paths reachable"
- "connect chosen path"
- "make output free"
- "output only station-user"
- "output not only station-user"

Als het aantal mogelijke paden groter dan nul is (reach_paths > 0) moet nu de beste mogelijkheid worden gekozen ("choose best"), "choose best" komt in de plaats van "take one". Verder moet het gedeelte "rearranging" worden aangepast, en tenslotte bij een vrijgave (call_release) moet het gedeelte "remove station from endswitch" worden gewijzigd.
Hieronder volgt het programma, de gedeelten tussen " " worden daarna besproken (voor zover ze gewijzigd zijn).

```
begin
    glovar call_request,call_release;
    glovar stat_req,outp_req,outp_rel;
    glocon M,n2;
    privar counter,x,max,pos;
    privar station,multipl;
    privar reach_paths;
    privar midswitch,secsw_input,endsw_input;
    privar always;
    always vir bool:=true;
    call_request vir bool,call_release vir bool:=false,false;
    "initialisation";
    do always ->
        "look for call_request and/or call_release";
        do call_request ->
            "inlet available ?";
            if "inlet available" ->
                "connect outlet"
            [] "inlet not available" ->
                "which paths reachable";
                if reach_paths > 0  ->
                    "choose best"
                [] reach_paths = 0  ->
```

```
                    "rearranging"
                fi;
                 "connect chosen path"
            fi;
            output(outp_req):=stat_req;
            call_request:=false
        [] call_release ->
            "make output free";
            if "output only station-user" ->
                "remove station from endswitch"
            [] "output not only station-user" ->
                skip
            fi;
            call_release:=false
        od
    od
end
```

"initialisation"

Om de methode van het aanstampen toe te kunnen passen is naast de al bestaande variabelen (die hier herhaald zijn) nog een extra variabele nodig (packcounter):

- stat_avail : variabel
  functie: het bijhouden van de op de eindswitch aanwezige stations
  input : type station
  output : type multiplicity of nil
  geinitialiseerd op nil
- inp_of_endsw : variabel
  functie: het bijhouden welke middenswitches vrij zijn
  input : type middenswitch
  output : element van {free,busy}
  geinitialiseerd op free
- pos_paths : variabel
  functie: opslaan van de vrije mogelijkheden
  input : type integer
  output : type multiplicity of nil
  geinitialiseerd op nil
- user_counter : variabel
  functie: tellen van het aantal gebruikers van bepaalde ingang van eindswitch
  input : type middleswitch
  output : type integer
  geinitialiseerd op 0
- output : variabel
  functie: aangeven met welk station betreffende abonnee (outp_rel) verbonden is
  input : type output
  output : type station of nil
  geinitialiseerd op nil

- packcounter : variabel
  functie: bijhouden hoeveel verbindingen er via een bepaalde multipliciteit gelegd zijn
  input : type multiplicity
  output : type integer
  geinitialiseerd op 0

Tijdens de initialisatie moeten de gegevens van de verdeling van de stations over de secondswitches opgeslagen worden in de volgende 'variabelen' (net als bij het eerste programma):

- stat_on_secsw: constant
  functie: aangeven op welke middenswitches station aanwezig is
  input : type station,multiplicity
  output : type middleswitch
  geinitialiseerd op verdeling stations
- inp_of_secsw : constant
  functie: aangeven op welke ingang van middenswitch het station aanwezig is
  input : type station,multiplicity
  output : type inp_secsw
  geinitialiseerd op verdeling stations

Als er een oproep (call-request) binnenkomt moet de variabele packcounter alleen aangepast worden als het station nog niet op de eindswitch aanwezig is. En bij een vrijgave (call-release) alleen als de output (outp_release) de enige gebruiker van het station is.
Dit houdt in dat de variabele packcounter aangepast moet worden bij "choose best", tijdens "rearranging" en bij "remove station from endswitch".

"choose best"

In dit gedeelte moet een zo verstandig mogelijke keuze gemaakt worden uit het aantal beschikbare mogelijkheden.
De variabele x doorloopt de waarden van 0 t/m (reach_paths - 1). Met de variabele max wordt de drukst bezette (mogelijke) multipliciteit bijgehouden.
Tijdens het if-statement wordt gekeken of de huidige waarde van max de drukst bezette multipliciteit is of dat de multipliciteit die met de variabele pos_paths(x) wordt aangegeven zwaarder belast is. Als de variabele max de drukst bezette multipliciteit is hoeft er niets gedaan te worden, anders moet de waarde van pos_paths(x) in max gezet worden.
Nadat zo alle mogelijke multipliciteiten zijn bekeken staat in de variabele max de drukst bezette (mogelijke) multipliciteit.
Nu moet alleen nog de variabele packcounter(max), die het aantal verbindingen aangeeft dat via de drukst bezette mogelijke multipliciteit is doorgeschakeld, met 1 worden verhoogd.

```
x:=0;
max:=pos_paths(x);
do x < (reach_paths - 1) ->
              x:=x+1;
              if packcounter(max) >= packcounter(pos_paths(x)) ->
                                        skip
              [] packcounter(max) < packcounter(pos_paths(x)) ->
                                        max:=pos_paths(x)
              fi
od;
multipl:=max;
packcounter(multipl):=packcounter(multipl)+1
```

"remove station from endswitch"

Als er geen gebruiker meer is voor het betreffende station dan moet dit station van de eindswitch verwijderd worden.

```
disconnectmid : functie : in middenswitch verbreken van station
                          en uitgang (=ingang eindswitch)
                input   : type middleswitch,inp_secsw
                output  : geen
```

De negatieve waarde van de variabele stat_avail geeft aan dat het betreffende station niet meer op de eindswitch aanwezig is en wordt deze input van de eindswitch vrij geboekt.
Tenslotte moet de variabele packcounter van de betreffende multipliciteit met 1 verlaagd worden.

```
multipl:=stat_avail(station);
secsw_input:=inp_of_secsw(station;multipl);
midswitch:=endsw_input;
disconnectmid(midswitch;secsw_input);
stat_avail(station):= -inf;
inp_of_endsw(endsw_input):=free;
packcounter(multipl):=packcounter(multipl)-1
```

"rearranging"

Ook in dit gedeelte moeten een aantal wijzigingen worden aangebracht. De volgende gedeelten zijn hetzelfde als in het tweede programma (zie paragraaf 4.2 voor de toelichting):

- "init rearranging"
- "put stat_req in patharray"
- "blocked by which stations"
- "put possibility in freearray"
- "put blocking in blockarray"
- "check on cycle"

De wijzigingen in de overige gedeelten hebben betrekking op een betere keuze en op het bijhouden van de variabele packcounter.

Het gedeelte "choose one and put in patharray" uit het tweede programma wordt vervangen door "choose best and put in patharray" waarbij een verstandige keuze wordt gemaakt. Evenzo wordt "other choice" vervangen door "other good choice".
Als het voorkeurs pad met verleggingen is gevonden wordt dit weer in "rear" gelegd.
Net als in het tweede programma wordt zolang de boolean variabele blocked true is een volgend station voor het pad gezocht. Ook hier wordt weer een stap terug gedaan als er een cycle is ontstaan.

Hieronder volgt het gedeelte "rearranging", de teksten tussen " " worden daarna besproken.

```
begin
   glocon M,n2,stat_req;
   glovar always,multipl,midswitch,secsw_input,endsw_input;
   glovar station,counter,x,max;
   privar blocked,cycle,end_of_path,secsw,candi;
   privar stat_itself,freepos,new_endsw_input,y;
   pricon cycle_min;
   "init rearranging";
   station:=stat_req;
   stat_itself:=nil;
   blocked:=true;
   freepos:=0;
   cycle:=false;
   "put stat_req in patharray";
   "blocked by which stations";
   do blocked ->
            if not cycle -> "choose best and put in patharray"
            [] cycle     -> "one step back";
                            "other good choice"
            fi;
            "check on cycle";
            if not cycle -> "blocked by which stations"
            [] cycle     -> skip
            fi
   od;
   "rear"
end
```

"choose best and put in patharray"

Net als bij "choose one and put in patharray" wordt nadat een station (tak in de tree) is gekozen deze multipliciteitswaarde en het station opgeslagen in de variabele patharray.
Alleen moet nu een verstandige keuze worden gemaakt, deze keuze gaat net zo als bij "choose best".

```
x:=0;
```

```
max:=0;
do x < M -> if x = stat_itself -> skip
            [] x <> stat_itself ->
                      if max >= packcounter(x) -> skip
                      [] max < packcounter(x) -> max:=packcounter(x)
                      fi
            fi;
            x:=x+1
od;
end_of_path:=end_of_path+1;
patharray(end_of_path):=x;
end_of_path:=end_of_path+1;
patharray(end_of_path):=blockarray(x);
station:=patharray(end_of_path)
```

"other good choice"

Hier wordt ook weer een verstandige keuze gemaakt waarbij net als in "other choice" bij het tweede programma de al bekeken mogelijkheden buiten beschouwing moeten worden gelaten. Hier wordt dat bereikt door de momentele waarde van de variabele x te vergelijken met de variabele triedmultipl(station;y) waarbij de variabele y behoort tot het interval [0,multicount(station)].
Nadat een verstandige keuze is gemaakt, wordt de multipliciteitswaarde en het blokkerende station opgeslagen in de variabele patharray.

```
x:=0;
max:=0;
do x < M ->
  if x = [triedmultipl(station;y) | 0<=y<=multicount(station))] ->
                                                        skip
  [] x <> [triedmultipl(station;y) | 0<=y<=multicount(station))] ->
              if max >= packcounter(x) -> skip
              [] max < packcounter(x) -> max:=packcounter(x)
              fi
  fi;
  x:=x+1;
od;
end_of_path:=end_of_path + 1;
patharray(end_of_path):=x;
end_of_path:=end_of_path + 1;
patharray(end_of_path):=blockarray(x);
station:=patharray(end_of_path)
```

"rear"

Dit gedeelte blijft vrijwel ongewijzigd ten opzichte van "rear" in het tweede programma, alleen moet ook hier de willekeurige keuze vervangen worden door een verstandige keuze.

De drukst bezette multipliciteit wordt weer in de variabele max opgeslagen en bijgehouden.

```
x:=0;
max:=freearray(x);
do x < (freepos - 1) ->
              x:=x+1;
              if packcounter(max) >= packcounter(freearray(x)) ->
                                        skip
              [] packcounter(max) < packcounter(freearray(x)) ->
                                        max:=freearray(x)
              fi
od;
multipl:=max;
packcounter(multipl):=packcounter(multipl)+1;
station:=patharray(end_of_path);
multicount(station):=0;
end_of_path:=end_of_path - 1;
do end_of_path >= 1 ->
   midswitch:=stat_on_secsw(station;multipl);
   secsw_input:=inp_of_secsw(station;multipl);
   connectmid(midswitch;secsw_input);
   counter:=stat_avail(station);
   endsw_input:=stat_on_secsw(station;counter);
   new_endsw_input:=midswitch;
   counter:=0;
   do counter < n2 ->
      if known(counter)     -> skip
      [] not known(counter) ->
         if output(counter) <> station  and output(counter) <> nil ->
                                        skip
         [] output(counter) = nil     -> known(counter):=true
         [] output(counter) = station -> known(counter):=true;
                                connectend(new_endsw_input;counter);
                                disconnectend(endsw_input;counter)
         fi
      fi;
      counter:=counter + 1
   od;
   midswitch:=endsw_input;
   secsw_input:=inp_of_secsw(station;triedmultipl(station;0));
   disconnectmid(midswitch;secsw_input);
   user_counter(new_endsw_input):=user_counter(endsw_input);
   user_counter(endsw_input):=0;
   stat_avail(station):=multipl;
   inp_of_endsw(endsw_input):=free;
   inp_of_endsw(new_endsw_input):=busy;
   multipl:=patharray(end_of_path);
   end_of_path:=end_of_path - 1;
   station:=patharray(end_of_path);
   multicount(station):=0;
   end_of_path:=end_of_path - 1
od;
```

Hiermee is het derde programma volledig. In het volgende hoofdstuk volgen de conclusies en aanbevelingen.

5 CONCLUSIES EN AANBEVELINGEN

In paragraaf 1 zullen de conclusies worden getrokken van dit afstudeerwerk. En in paragraaf 2 worden dan nog aanbevelingen gedaan hoe dit werk voortgezet zou kunnen worden.

5.1 Conclusies

In dit verslag is de besturing van het meest effectieve distributie netwerk dat tot op heden bekend is (= het Richards netwerk) beschreven. Met de 'taal' van Dijkstra is het goed mogelijk deze besturing te beschrijven.
Het vermoeden bestaat dat door gebruik te maken van de strategie aanstampen, het aantal verleggingen beperkt kan blijven.
In het Richards netwerk kunnen zich geen cycles (stations die meerdere keren voorkomen in een pad) ter lengte twee voordoen.
Een bepaalde verbinding hoeft tijdens het verleggen hooguit maar een keer verlegd te worden.

Door op elke vertakking van de tree de meest belaste multipliciteit te kiezen is waarschijnlijk de kans het grootst dat op een niveau lager een vrije mogelijkheid wordt gevonden.
Of uniform zoeken over de tree sneller resultaat geeft dan het kiezen van de zwaarst belaste multipliciteit zal door simulaties moeten worden bepaald.

5.2 Aanbevelingen

Om meer zekerheid te hebben of aanstampen wel de juiste strategie is om het aantal verleggingen te beperken zullen er simulaties uitgevoerd moeten gaan worden waarbij aanstampen dan tegen de andere methoden getoetst wordt.
Verder zou het misschien beter zijn geweest als meer aan de functiestructuur was vastgehouden in plaats van zoals nu is gedaan, de functies meteen uitvoeren als arrays. Achteraf is het duidelijk waardoor dit werd veroorzaakt, namelijk door het assignment statement. Hierdoor kan een functie geen functie meer blijven maar moet dan geimplementeerd worden als een array.

LITERATUURLIJST

1. C.Clos: A Study of Non-Blocking Switching Networks, Bell Syst. Tech. J., Vol. 32, 1953, 406-424 (no.2)
2. C.M.Melas: System for Rearranging Paths in a Blocking Switching Network, European Patent Specification, Publ.nr. 0073917 B1 Publ.date 091085, Appl.nr. 82106650.3, Int.cl. H04 Q 3/68, H04 Q 3/52.
3. F.K.Hwang en A.Jajszczyk: On Nonblocking Multiconnection Networks, to be published in Bell Syst. Tech. J.
4. G.M.Masson: Binomial Switching Networks for Concentration and Distribution, IEEE Trans. Commun., Vol. COM-25, 1977, 873-883
5. J.C.A.Boekhorst: Switching network structures for dialogue and distribution traffic (Ned.), APT-rapport, JNL 206-RO-346/Expl.
6. G.W.Richards en F.K.Hwang: A Two-Stage Rearrangeable Broadcast Switching Network, IEEE Trans. Commun., Vol. COM-33,1985, 1025-1035.
7. E.W.Dijkstra: A discipline of programming, New Jersey, Prentice-Hall, 1976
8. Vakgroep Automatische Verkeerssystemen, college dictaat Automatische Telefonie, Delft, 1984.
9. Vakgroep Automatische Verkeerssystemen, college dictaat Electronische Automatische Telefonie, Delft, 1985.
10. S.Nakamura en G.M.Masson: Lower Bounds on Crosspoints in Concentrators, IEEE Trans. on Computers, Vol. C-31, dec. 1982, 1173-1179.

BIJLAGE 1 : De bewijzen van de stellingen

In deze bijlage worden de bewijzen gegeven van de in dit verslag genoemde stellingen.

Stelling 1 :
Bij het Richards netwerk kan zich geen cycle ter lengte twee voordoen.

Bewijs :
Door voor n1 (=aantal ingangen per middenswitch) een priemgetal te nemen komt elk tweetal stations maar op een middenswitch samen voor. Dus kan een station een ander station maar op een middenswitch blokkeren en dus zijn er voor een cycle (=station dat meerdere keren voorkomt in een pad) minimaal drie stations nodig.

Stelling 2 :
In een herarrangeerbaar algemeen Masson netwerk hoeft voor een verbindingsaanvrage een bestaande verbinding nooit meer dan eens te worden verlegd.

Bewijs : Het bewijs volgt direct uit het bewijs van de volgende (hulp-) stelling.

Hulp-Stelling : Een algemeen Masson netwerk dat herarrangeerbaar is, is ook sequentieel herarrangeerbaar.

Bewijs : Laat B de aanduiding zijn van een interne netwerktoestand, die blokkerend is voor een bepaalde verbindingsaanvrage. Aangezien het netwerk herarrangeerbaar is, bestaan er ook netwerktoestanden die dezelfde externe verbindingen implementeren als B, maar waarvoor de aanvrage in kwestie niet blokkerend is. Laat N de aanduiding zijn van zo een netwerktoestand (keuze willekeurig).

In het algemene Masson netwerk is nu een middenswitch aan te wijzen die vrij is in de toestand N en die belegd is in de toestand B. Immers, de aanvrage is in toestand N direct te leggen, maar in B niet. Omdat B en N dezelfde externe verbindingen implementeren, is het aantal belegde middenswitches in beide toestanden gelijk. Dus is er ook een middenswitch aan te wijzen die vrij is in B en belegd in N. Kennelijk is het mogelijk in toestand B een verlegging uit te voeren, zodanig dat laatst genoemde middenswitch door hetzelfde station belegd wordt als in toestand N.

Zolang de interne netwerktoestand blokkerend is voor de aanvrage duiden we deze toestand (opnieuw) aan met B en voeren we de verlegging uit beschreven in de vorige alinea. Dit procede zal zeker eindigen aangezien het aantal middenswitches dat door hetzelfde station wordt belegd als in toestand N, door de verlegging, toeneemt (met 1) en het aantal middenswitches eindig is. Na afloop van het procede is de toestand van het netwerk niet langer blokkerend voor de aanvrage.

Uit dit bewijs volgt tevens dat de uitgevoerde verleggingen steeds betrekking hebben op verschillende stations, zodat een bestaande verbinding nooit meer dan eens hoeft te worden verlegd.

Vermoeden 2:

Door bij elke vertakking van de tree, als er nog geen vrije mogelijkheid is gevonden, de zwaarst belaste (=drukst bezette) multipliciteit te kiezen, is de kans het grootst dat op een niveau lager een vrije mogelijkheid wordt gevonden.

Toelichting :

Deze methode van het zoeken van een pad toont veel overeenkomst met de methode van het leggen van de oproepen, ook daar wordt de meest belaste multipliciteit gekozen.

Door bij een vertakking van de tree de meest belaste multipliciteit te kiezen wordt op een niveau lager de keuze bij de volgende vertakking van de tree gemaakt uit de overige multipliciteiten (=alle behalve de drukst bezette). De kans op een vrije mogelijkheid zal hierdoor groter zijn dan bij een andere keuze.

BIJLAGE 2 : Uitgewerkte voorbeelden

In deze bijlage worden voor de multipliciteiten 2 en 3 enige voorbeelden gegeven. Aan de hand van dit soort voorbeelden is de keuze gemaakt tussen de besturingsstrategie aanstampen en uniform over de multipliciteiten leggen van de oproepen.
Voor M=2 geldt volgens het artikel van Richards en Hwang dat het aantal stations N1 kleiner dan 35 moet zijn, hier wordt N1=25 genomen. Het maximale aantal uitgangen per eindswitch is 5 (n2), dit houdt in dat er 5 oproepen doorgeschakeld moeten kunnen worden.

Hieronder staat een voorbeeld waarbij de stations :8,18,16,20 en 7 doorgeschakeld moeten worden. Als de stations in deze volgorde aangeboden worden kan station 7 niet gelegd worden, er zal dan eerst verlegd moeten worden.

		doorgeschakeld station
	1 2 3 4 5	
	6 7 8 9 10	8
0	11 12 13 14 15	
	16 17 18 19 20	18
	21 22 23 24 25	
- - - - - - - - - - - - -		
	1 22 18 14 10	
	6 2 23 19 15	
1	11 7 3 24 20	20
	16 12 8 4 25	16
	21 17 13 9 5	

Bij multipliciteit 2 is er geen verschil gevonden tussen uniform leggen en aanstampen.

Voor M=3 ligt het aantal stations tussen 34 en 117 in (volgens het artikel van Richards en Hwang). In de onderstaande voorbeelden is voor N1=49 gekozen. Het maximale aantal uitgangen per eindswitch is 13.

eerste voorbeeld:
In dit voorbeeld moeten de volgende stations achtereenvolgens gelegd worden :

34,10,28,45,19,23,48,30,43,39,12,1

Het station 12 is niet te leggen, eerst zijn er verleggingen nodig. In dit voorbeeld is er geen verschil tussen uniform leggen en aanstampen.

		doorgeschakeld station
	1 2 3 4 5 6 7	
	8 9 10 11 12 13 14	10
	15 16 17 18 19 20 21	19
0	22 23 24 25 26 27 28	28
	29 30 31 32 33 34 35	34
	36 37 38 39 40 41 42	39
	43 44 45 46 47 48 49	45
- - - - - - - - - - - - - - -		
	1 44 38 32 26 20 14	
	8 2 45 39 33 27 21	
	15 9 3 46 40 34 28	
1	22 16 10 4 47 41 35	
	29 23 17 11 5 48 42	23
	36 30 24 18 12 6 49	30
	43 37 31 25 19 13 7	43
- - - - - - - - - - - - - - -		
	1 37 24 11 47 34 21	
	8 44 31 18 5 41 28	
	15 2 38 25 12 48 35	48
2	22 9 45 32 19 6 42	
	29 16 3 39 26 13 49	
	36 23 10 46 33 20 7	
	43 30 17 4 40 27 14	

tweede voorbeeld :
Hier moeten de stations :

1,10,19,28,34,39,45,17,48,30,40,43,12

gelegd worden. Voor aanstampen kan station 12 niet meer doorgeschakeld worden, terwijl dit voor uniform leggen nog wel mogelijk is.

		aanstampen	uniform leggen
	1 2 3 4 5 6 7	1	1
	8 9 10 11 12 13 14	10	10
	15 16 17 18 19 20 21	19	19
0	22 23 24 25 26 27 28	28	28
	29 30 31 32 33 34 35	34	34
	36 37 38 39 40 41 42	39	39
	43 44 45 46 47 48 49	45	45
- - - - - - - - - - - - - - - - - - -			
	1 44 38 32 26 20 14		
	8 2 45 39 33 27 21		
	15 9 3 46 40 34 28	40	40
1	22 16 10 4 47 41 35		
	29 23 17 11 5 48 42	17	17
	36 30 24 18 12 6 49	30	12
	43 37 31 25 19 13 7	43	43
- - - - - - - - - - - - - - - - - - -			
	1 37 24 11 47 34 21		
	8 44 31 18 5 41 28		
	15 2 38 25 12 48 35	48	48
2	22 9 45 32 19 6 42		
	29 16 3 39 26 13 49		
	36 23 10 46 33 20 7		
	43 30 17 4 40 27 14		30

derde voorbeeld :
Tijdens dit voorbeeld blijkt dat voor uniform leggen niet alle gevraagde stations te leggen zijn (station 19 is niet te leggen), voor aanstampen is dit wel mogelijk.
De gevraagde stations zijn :

8,12,17,25,29,35,31,40,49,10,30,9,19.

		aanstampen	uniform leggen
0	1 2 3 4 5 6 7		
	8 9 10 11 12 13 14	8	8
	15 16 17 18 19 20 21	17	17
	22 23 24 25 26 27 28	25	25
	29 30 31 32 33 34 35	29	29
	36 37 38 39 40 41 42	40	40
	43 44 45 46 47 48 49	49	49
1	1 44 38 32 26 20 14		
	8 2 45 39 33 27 21		
	15 9 3 46 40 34 28	9	
	22 16 10 4 47 41 35	35	35
	29 23 17 11 5 48 42		
	36 30 24 18 12 6 49	12	12
	43 37 31 25 19 13 7	31	31
2	1 37 24 11 47 34 21		
	8 44 31 18 5 41 28		
	15 2 38 25 12 48 35		
	22 9 45 32 19 6 42	19	9
	29 16 3 39 26 13 49		
	36 23 10 46 33 20 7	10	10
	43 30 17 4 40 27 14	30	30

vierde voorbeeld :
Hier moeten de stations

3,11,15,27,38,45,20,13,16,26,49,12,10

doorgeschakeld worden.
Vanaf het moment dat station 26 gelegd moet worden, moet er met de methode van uniform leggen verlegd worden. Met de methode aanstampen is het leggen van deze stations tot de capaciteit mogelijk.

		aanstampen	uniform leggen
	1 2 3 4 5 6 7	3	3
	8 9 10 11 12 13 14	11	11
	15 16 17 18 19 20 21	15	15
0	22 23 24 25 26 27 28	27	27
	29 30 31 32 33 34 35		
	36 37 38 39 40 41 42	38	38
	43 44 45 46 47 48 49	45	45
- - - - - - - - - - - - - - - - - - - -			
	1 44 38 32 26 20 14	20	20
	8 2 45 39 33 27 21		
	15 9 3 46 40 34 28		
1	22 16 10 4 47 41 35	16	16
	29 23 17 11 5 48 42		
	36 30 24 18 12 6 49	49	
	43 37 31 25 19 13 7	13	
- - - - - - - - - - - - - - - - - - - -			
	1 37 24 11 47 34 21		
	8 44 31 18 5 41 28		
	15 2 38 25 12 48 35	12	
2	22 9 45 32 19 6 42		
	29 16 3 39 26 13 49	26	13
	36 23 10 46 33 20 7	10	
	43 30 17 4 40 27 14		

Hieronder volgt nog een toelichting op de methode uniform leggen.
Bij uniform leggen wordt, voordat een station gelegd wordt, gekeken voor de multipliciteiten 0,...,M-1 (voor de bovenstaande voorbeelden 0,1 en 2) hoeveel mogelijkheden de stations over houden als het station over de betreffende multipliciteit gelegd wordt.
Voor het vierde voorbeeld volgt hieronder de keuze bepaling voor het leggen van station 45.

Op multipliciteit 0 blokkeert station 45	station 43	: 2 multipl. over
	44	: 2
	45	: zelf
	46	: 2
	47	: 2
	48	: 2
	49	: 2
Op multipliciteit 1 blokkeert station 45	station 8	: 1 multipl. over
	2	: 1
	45	: zelf
	39	: 1
	33	: 2
	27	: 1
	21	: 1
Op multipliciteit 2 blokkeert station 45	station 22	: 1 multipl. over
	9	: 1
	45	: zelf
	32	: 2
	19	: 1
	6	: 1
	42	: 1

Hieruit blijkt dat de beste keuze multipliciteit 0 is, omdat dan voor elk van de betrokken stations evenveel mogelijkheden over blijven.
Zo kan elke keer de keuze bepaald worden, soms komt het ook voor dat er tussen twee multipliciteiten geen verschil is, dan is de keuze daartussen willekeurig.

BIJLAGE 3 : De eindbespreking

De eindbespreking van het afstudeerwerk werd gehouden op 17 februari.
De examencommissie bestond uit :

ing. J. van Baardewijk	APT-Hilversum
ir. R.A. Beukers	TU-Delft
ir. J.C.A. Boekhorst	APT-Hilversum
prof. ir. J.L. de Kroes	TU-Delft (voorzitter)

In deze bijlage worden de op- en aanmerkingen op het afstudeerverslag behandeld voor zover ze niet in het verslag zelf zijn verwerkt.

1. Een algemene opmerking werd gemaakt dat er in het verslag harde uitspraken worden gedaan die niet onderbouwd zijn, bv. het Clos netwerk is het beste (welke criteria ?).

2. Er werd gevraagd of random leggen niet uniformer is dan de in dit verslag genoemde methode uniform leggen.
   De methode uniform leggen probeert het aantal mogelijkheden voor elk station zo gelijkmatig mogelijk te houden, dus voor elk station een gelijk aantal vrije mogelijkheden.

3. Masson noemt zijn concentrator optimaal (zie ook blz. 12), doordat Masson gebruik maakt van een binomiale concentrator moet er een bepaald verband bestaan tussen het aantal inputs, aantal outputs en de capaciteit.

4. Volgens de CCITT is de benaming zoals die in dit verslag gebruikt wordt ten aanzien van de connectietrap fout. De connectietrap schakelt niet en is, volgens de CCITT, daarom geen trap.

5. Tijdens voorbeelden zijn alleen de methoden aanstampen en uniform leggen bekeken. Een alternatief zou zijn cyclisch over de multipliciteiten te leggen. Simulaties zullen moeten uitwijzen welke methode beter is.

6. Tijdens onderzoek uitgegaan van de eindswitch. Voor praktische uitvoering (met micro-processor) ook de middentrap meenemen.

7. Het verleggen heeft 2 aspecten : niet alleen in de middentrap zijn verleggingen nodig, ook in de eindtrap !
   Bij de stellingen worden alleen de verleggingen in de middentrap bedoeld.

8. Als mogelijkheid om het aantal keren dat er verlegd moet worden ontbreekt de adaptieve methode. Aangezien er geen gegevens over de mogelijke stations/video-diensten was is deze methode weggelaten. Tijdens simulaties moet deze methode zeker meegenomen worden.

9. De array-variabelen waren oorspronkelijk bedoeld als functies, maar door het assignment-statement was een interpretatie als functie niet meer mogelijk.

10. De stap van het algemene Masson netwerk naar het Richards netwerk bestaat ondermeer uit :
    - invoering van multipliciteiten.
    - elk tweetal stations maar op 1 middenswitch samen voor.