ORGANISATIE VAN BEDRIJFSSYSTEMEN VOOR COMPUTER NETWERKEN

Ingrid J.M. Birkhoff en Jan van Leeuwen

RUU-CS-81-14

Augustus 1981

Rijksuniversiteit Utrecht

Vakgroep informatica

Princetonplein 5
Postbus 80.002
3508 TA Utrecht
Telefoon 030–531454
The Netherlands

ORGANISATIE VAN BEDRIJFSSYSTEMEN VOOR COMPUTER NETWERKEN

Ingrid J.M. Birkhoff en Jan van Leeuwen

Technical Report RUU-CS-81-14

Augustus 1981

Department of Computer Science
University of Utrecht
P.O. Box 80.002, 3508 TA Utrecht
the Netherlands

ORGANISATIE VAN BEDRIJFSSYSTEMEN VOOR COMPUTER NETWERKEN.

Ingrid J.M. Birkhoff en Jan van Leeuwen.
Vakgroep Informatica, R.U.Utrecht
postbus 80.002, 3508 TA Utrecht

I. Inleiding.

Een computer netwerk bestaat uit een aantal autonome hosts en een communicatie subnet. De hosts zijn gewone machines waarop gebruikersprocessen verwerkt kunnen worden. Het subnet bestaat uit intermediate message processors (IMP's) en communicatielijnen. Het transport van boodschappen tussen de hosts wordt door het subnet verzorgd.

Er zijn twee typen boodschappen: messages en control messages. Messages bevatten willekeurige informatie die gebruikers (processen) willen verzenden. Control messages bevatten specifieke informatie, welke ten behoeve van het ordelijk functioneren van het systeem verzonden moet worden.

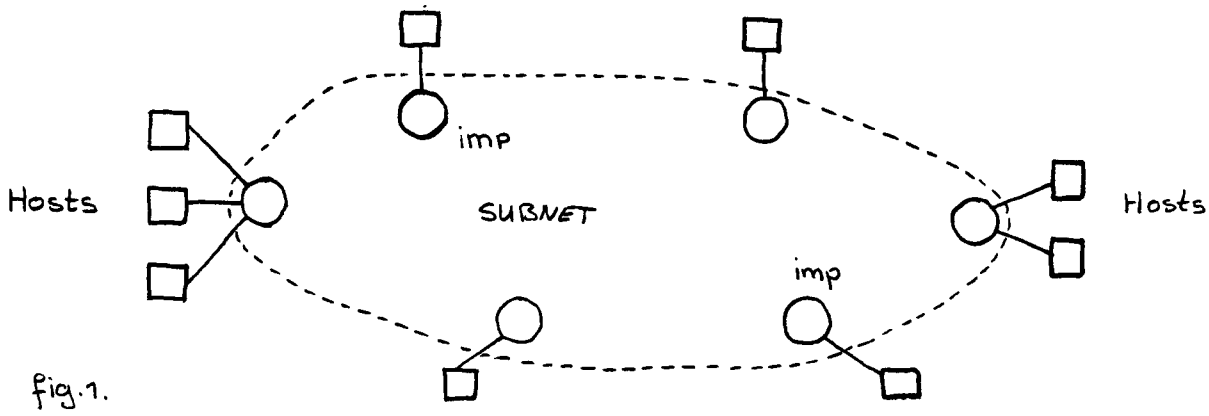


fig.1.

De hosts zijn via de IMP's met het subnet verbonden (fig.1). Wanneer een host een boodschap naar een andere host verstuurd wil hebben, levert hij hem af bij de IMP waarmee hij aan het subnet verbonden is. Eventueel via een aantal tussenliggende IMP's, bereikt de boodschap de IMP waarmee de host die de bestemming is , met het subnet verbonden is. Elke IMP op de route van het transport slaat de boodschap op, en verzendt hem wanneer de benodigde outputlijn vrij is. Een netwerk dat deze techniek gebruikt wordt een store-and-forward netwerk genoemd. Worden messages als een geheel behandeld en verstuurd, dan spreekt men van message-switching.

Omdat de boodschappen willekeurig lang kunnen zijn, moet elke IMP een, in principe, onbeperkte bufferruimte hebben. In een packet-switching netwerk worden boodschappen verdeeld in packets met een voorgeschreven maximale lengte. Deze maximale packetlengte kan zodanig gekozen worden dat de beschikbare bufferruimte bij elke IMP voldoende is. Een packet-switching netwerk voert transporten sneller uit

dan een message-switching netwerk doordat van messages die uit meerdere packets bestaan het eerste packet al door een IMP verstuurd kan worden voordat de volgende packets bij die IMP gearriveerd zijn.

In dit rapport wordt beschreven welke taken een bedrijfssysteem voor een computernetwerk moet uitvoeren, en hoe dergelijke bedrijfssystemen georganiseerd zijn.

II. Taken van een networkwide operating system.

De taken van een networkwide operating system (NWOS) bestaan uit:
1. Het bieden van een interface met alle resources van het netwerk,
2. Het beheer voeren over die resources,
3. Het beheer van het subnet.

II.1 Het interface met de gebruikers.

Het NWOS vormt het interface tussen de gebruikers en het netwerk. In principe moet het NWOS de gebruikers een uniform interface met het netwerk bieden, dat alle (fysische) verschillen tussen de locale hosts en tussen de resources verbergt. Een gebruiker zou nooit behoeven te weten, of een resource dat hij gebruikt, een locale of een remote resource is.

Om een dergelijk ideaal interface met de resources van het netwerk te verkrijgen, moet het NWOS de gebruikers een Networkwide Control Language (NCL) en een Networkwide File System (NFS) bieden. De NCL kan een hogere programmeertaal zijn. Programma's in NCL worden geexecuteerd door een (virtuele) machine. Deze machine voert de remote access procedures uit. Het gebruik van een NCL voorkomt dat de gebruikers alle control languages van alle hosts waarmee ze ooit eens werken, zullen moeten kennen, en dus ook ten allen tijde moeten weten met welke host ze werken. Wanneer de gebruikers de beschikking krijgen over een NFS, behoeven zij de locale file systems niet meer te kennen. File access kan dan, onafhankelijk van de locatie van de files, altijd op dezelfde manier geschieden.

Het NWOS moet de gebruiker wel de mogelijkheid laten om, wanneer hij dat wenst, informatie over de locatie van de resources te verkrijgen, of zelf de locatie van een resource te specificeren.

II.2 Resource management.

De resources van het netwerk zijn: de processoren van de locale hosts, files, databases en I/O devices.

A. Processor management.

De processor manager verzorgt de toewijzing van de processoren uit het netwerk aan processen die daar om verzocht hebben. De locatie van de host waar de toegewezen processor zich bevindt, vormt nu de bestemming van het proces. Als er meerdere processen tegelijk het netwerk binnen komen en om een processor verzoeken, dan zal de processor manager ze een processor toewijzen, die mogelijk niet de processor van de host is waar het proces het netwerk binnen kwam. Dit gebeurt bij

voorbeeld wanneer er bij een van de hosts een job ingelezen wordt die uit meerdere parallelle processen bestaat, of er meer jobs aangeboden worden dan de host verwerken kan, of indien een proces om faciliteiten vraagt die de host zelf niet kan bieden.

Bij het zoeken naar een optimale scheduling van de processen over de processoren dient de processor manager rekening te houden met de volgende factoren:
- de capaciteit, en de huidige bezetting van de processoren,
- de lengte van de processen,
- de kosten en duur van het transport van een proces naar en van de toegewezen processor,
- een eventueel voorgeschreven executievolgorde van een serie processen,
- de mogelijkheid om eventueel pre-emption te plegen.

Zij $c(i)$ de voltooiingstijd van proces i bij een bepaalde scheduling. En definieer $d(i)$ als de (gewenste) afleveringstijd. Dan is $l(i)=c(i)-d(i)$ de laatheid, en $t(i)=\max\{0,l(i)\}$ de traagheid van proces i. Zij $u(i)=0$ als $l(i)<=0$, $u(i)=1$ als $l(i)=0$. De scheduling van n processen over m machines kan nu geoptimaliseerd worden ten aanzien van een van de volgende grootheden:
- de totale voltooiingstijd $C_{max}=\max\{c(i):i=1,\ldots,n\}$,
- de som van de voltooiingstijden $\Sigma c(i)$ van de n processen,
- de gewogen som van de voltooiingstijden $\Sigma w(i)*c(i)$; voor $w(i)=1/n$ is dat de gemiddelde voltooiingstijd van een proces.
- $L_{max}=\max\{l(i):i=1,\ldots,n\}$ de maximale laatheid van een proces,
- de totale traagheid $\Sigma t(i)$.
- de gewogen traagheid $\Sigma w(i)*t(i)$.
- het totale, gemiddelde en gewogen aantal te late processen $\Sigma u(i)$, $1/n*\Sigma u(i)$, $\Sigma w(i)*u(i)$.

Het optimaliseren van andere grootheden zoals idle time, bezetting van de processoren of wachttijd van de processen is equivalent aan optimalisatie van een van de bovenstaande grootheden (Rinnooy Kan[17]).

De meeste van de bovenstaande scheduling problemen zijn NP-volledig.

Bij de bestudering van scheduling problemen wordt er steeds van uitgegaan dat alle processoren initieel onbezet zijn. In een netwerk zal dit over het algemeen niet het geval zijn. Bij het zoeken naar een optimale scheduling moet de processor manager ook rekening houden met de heersende bezetting van de processoren. Misschien zou de processor manager zelfs kunnen besluiten om een in executie zijnd proces te stoppen, en elders de executie te laten hervatten. Eventuele transportkosten moeten dan wel bij zo'n besluit meetellen.

B. File management.

De netwerk file manager moet zowel de opslag van nieuw gecreëerde, als het access tot de bestaande files verzorgen.

Wanneer de gebruiker zelf de locatie heeft gespecificeerd waar hij zijn file opgeslagen wil hebben, dan behoeft de file manager alleen maar te controleren of er op die locatie voor die gebruiker voldoende ruimte beschikbaar is.

Voor files waarvan de gebruiker de locatie voor opslag niet aangegeven heeft, moet de file manager een geschikte plaats zoeken. Daarbij moet rekening gehouden worden met:
- de kosten van de opslag op een bepaalde locatie, en van het transport naar die locatie toe,
- de omvang van de betreffende file, en de beschikbare ruimte op de locatie,

- de tijd die in de toekomst nodig zal zijn om access tot de file te verkrijgen.

Mogelijk kan de file manager zelfs besluiten om reeds bestaande files te verplaatsen, bijvoorbeeld wanneer er op een gespecificeerde locatie niet genoeg ruimte is.

C. Database management.

De onderdelen van een database kunnen over een aantal locaties in het netwerk verspreid zijn. Voorts kunnen er van bepaalde onderdelen copieen bestaan.

Laten we het kleinste onderdeel van een database een "eenheid" noemen, en veronderstel dat een database uit een eindig aantal eenheden bestaat. Elke eenheid heeft een waarde.

Een database heet consistent wanneer de waarden van de eenheden aan bepaalde consistentie eisen voldoen. Dit kunnen eisen zijn zoals: het aantal gereserveerde plaatsen op een rondvaartboot is niet groter dan het aantal beschikbare plaatsen, of het negatieve saldo blijft binnen het toegestane crediet. Voor eenheden waarvan copieen bestaan, is er altijd de eis dat de copieen identiek zijn.

Een concurrency control is een algorithme, dat de verzoeken van gebruikersprocessen behandelt om een eenheid van de database te lezen of te beschrijven. Er kunnen meerdere verzoeken voor lezen en/of schrijven tegelijk gedaan worden. Er zal naar gestreefd moeten worden zoveel mogelijk verzoeken parallel af te werken. Het concurrency control zal hiertoe de verzoeken verstandig moeten schedulen. Daarbij moet het er voor zorgen dat elk proces een consistente database 'ziet', en na terminatie een consistente database achterlaat. Voorts moet het concurrency control er voor zorgen dat alle processen in eindige tijd termineren. Een goed concurrency control maximaliseert het aantal (parallelle) accesses tot de database, met handhaving van de consistentie. Hierbij moet opgemerkt worden dat het concurrency control normaal onafhankelijk is van de specifieke consistentieeisen voor de database.

Een ander probleem, dat de database manager moet oplossen, ontstaat wanneer een machine waar een deel van de database zich bevindt uitvalt. Als deze machine weer in bedrijf komt, moet het deel van de database zodanig hersteld worden dat weer aan de consistentie eisen voldaan wordt.

Op grond van veelvuldig access tot bepaalde delen van de database kan het gewenst zijn, een extra copie te creeren, of een deel van de database te verplaatsen naar een andere locatie. De database manager moet dan bepalen waar een nieuwe copie komt, of wat de nieuwe locatie gaat kosten. Daarbij moet rekening gehouden worden met transport- en opslagkosten.

Het vinden van een optimale oplossing voor vragen zoals "waar plaats je welk deel van de database?", blijkt vaak NP-volledig te zijn (zie Mahmoud[14], Srinivasan[20]). Het oplossen van problemen die in het algemeen ontstaan bij het creeren en uitbreiden van een database behoort niet tot de taken van de database manager.

D. Device management.

Een van de voordelen van het gebruik van een netwerk is, dat I/O-devices die maar op weinig locaties aanwezig zijn, zoals bijvoorbeeld fototypesetters, voor een veel grotere groep gebruikers beschikbaar komen.

De device manager verzorgt het toewijzen van devices aan processen. Wanneer een gebruiker zelf gespecificeerd heeft op welke locatie

hij het benodigde device wil gebruiken, behoeft de device manager alleen maar na te gaan of dat ook mogelijk is. Heeft de gebruiker de locatie niet zelf aangegeven, dan moet de device manager een geschikte locatie uitzoeken. Daarbij dient rekening gehouden te worden met:
- de bezetting en de capaciteit van de in aanmerking komende devices,
- de duur en de kosten van het transport van het proces naar de locatie van het toegewezen device,
- de eventuele extra belasting van de locale host.

II.3 Subnet management.

In het netwerk vinden transporten van control messages en messages plaats. De messages kunnen data, procescode of eventueel 'mail' bevatten.

Control messages bevatten informatie voor het systeem over, bijvoorbeeld, de toestand van het subnet, de status van de processen en de bezetting van de locale hosts. Datatransporten ontstaan vooral bij het gebruik van gedistribueerde databases. Wanneer gebruikers toegang willen hebben tot informatie die bij een remote host opgeslagen is, kan dat leiden tot transport van grote hoeveelheden data. De scheduling van processen door de proces manager leidt tot transporten van procescodes. Bij het versturen van mail naar andere gebruikers van het netwerk, wordt het netwerk als een communicatiemedium gebruikt dat zowel op de telefoon als op de post lijkt.

Laten we er van uit gaan dat we met een packet-switching netwerk te maken hebben. Dan worden dus alle messages in packets verdeeld getransporteerd.

De taak van de subnet manager bestaat uit het bepalen van een route voor elk packet, zodanig dat de packets hun nu bekende bestemming in eindige tijd bereiken.

Een optimaal routerings algorithme levert een zo groot mogelijk aantal packets zo snel mogelijk af op hun bestemming. Deze eisen zijn tegenstrijdig; een verhoging van de throughput zal een gemiddeld langer delay tot gevolg hebben. Een andere grootheid om te optimaliseren is het aantal IMP's dat een packet op zijn route bezoekt (het aantal 'hops' dat een packet maakt). Wanneer dit aantal hops klein is zullen packets, in het algemeen, snel (en goedkoop) hun bestemming bereiken.

Voorts moet het routerings algorithme de kans op een bottleneck (congestion) in het netwerk zo klein mogelijk houden. Dat kan door de transporten zoveel mogelijk te spreiden over de communicatielijnen. Het routerings algorithme kan congestion niet echt voorkomen want het kan bij een groot aanbod van packets geen packets weigeren. Daarom is er een apart congestion control algorithme nodig.

Routerings algorithmen kunnen in twee klassen worden verdeeld: adaptieve en niet-adaptieve algorithmen.

Adaptieve routerings algorithmen bepalen de route van een packet aan de hand van metingen of schattingen over de, op het moment heersende verkeersstroom in, en toestand van, het netwerk. Daartoe moet het algorithme op gezette tijden informatie krijgen over bijvoorbeeld de lengte van de wachtrijen, en de toestand van de IMP's en communicatielijnen. Met deze gegevens kan het algorithme dan weer betere routes bepalen voor de packets omdat het kan inspelen op veranderingen in het netwerk. Een goed adaptief routerings algorithme reageert snel op wijzigingen in de topologie van het netwerk. In ieder geval zo snel dat nieuwe routeringstabellen gecreeerd zijn voordat de topologie opnieuw verandert.

Niet-adaptieve routerings algorithmen gebruiken vaste routeringstabellen. Daardoor zijn ze op geen enkele wijze in staat om de strategie voor het bepalen van een optimale route aan te passen aan de

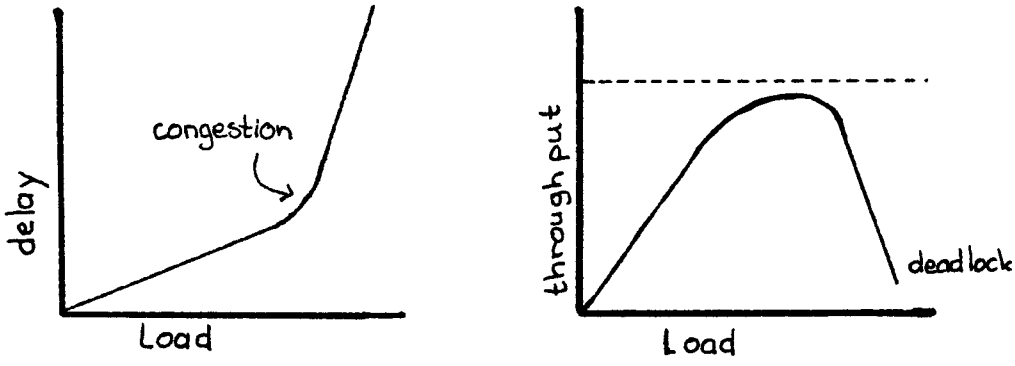
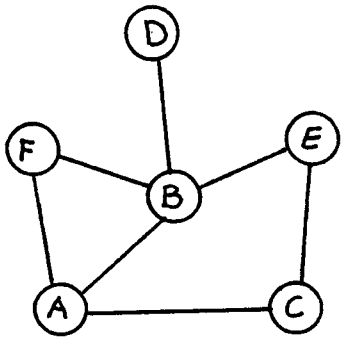


fig.2. congestion en deadlock.

zich wellicht steeds wijzigende toestand van het netwerk. Vooral datatransporten hebben een 'bursty' karakter: in een korte tijd worden er tussen twee hosts (IMP's) erg veel packets verzonden, waarna er een lange tijd niets meer volgt. Een niet-adaptief routerings algorithme is in zo'n geval niet in staat om eventuele bottlenecks te signaleren, en zich daaraan aan te passen.

Voorbeeld van een niet-adaptief routerings algorithme (static routing, zie[22]). Elke host H heeft een tabel waarin voor elke andere host G in het netwerk de beste, en op een na beste keuze staat voor de outputlijn van een packet met bestemming G. Bij elke keuze staat een gewicht w, hetgeen de kans is waarmee die outputlijn gekozen wordt.



Netwerk met zes hosts.

Routeringstabel voor A

bestemming	1e keus	2e keus
B	B 0.9	F 0.1
C	C 0.8	B 0.2
D	B 0.7	F 0.3
E	C 0.7	F 0.3
F	F 0.9	B 0.1

fig.3.

In figuur 3 wordt bijvoorbeeld een packet uit A met bestemming E met kans 0,7 via C, en met kans 0,3 via F, naar E getransporteerd.

De routes en gewichten zijn bepaald aan de hand van schattingen over de gemiddelde verkeersintensiteit in het netwerk. De routeringstabellen kunnen niet veranderd worden.

II.4 De architectuur van het netwerk.

De opbouw van de meeste computer netwerken is te zien als een

verzameling levels, waarbij elk level door een lager level wordt ondersteund en aan het volgende hogere level bepaalde faciliteiten biedt. Op elk level communiceren processen van verschillende hosts met elkaar volgens de regels die het protocol van het betreffende level voorschrijft. De International Standards Organisation (ISO) heeft een netwerkmodel ontworpen dat zeven levels telt (zie fig.4 en Tanenbaum[22]).

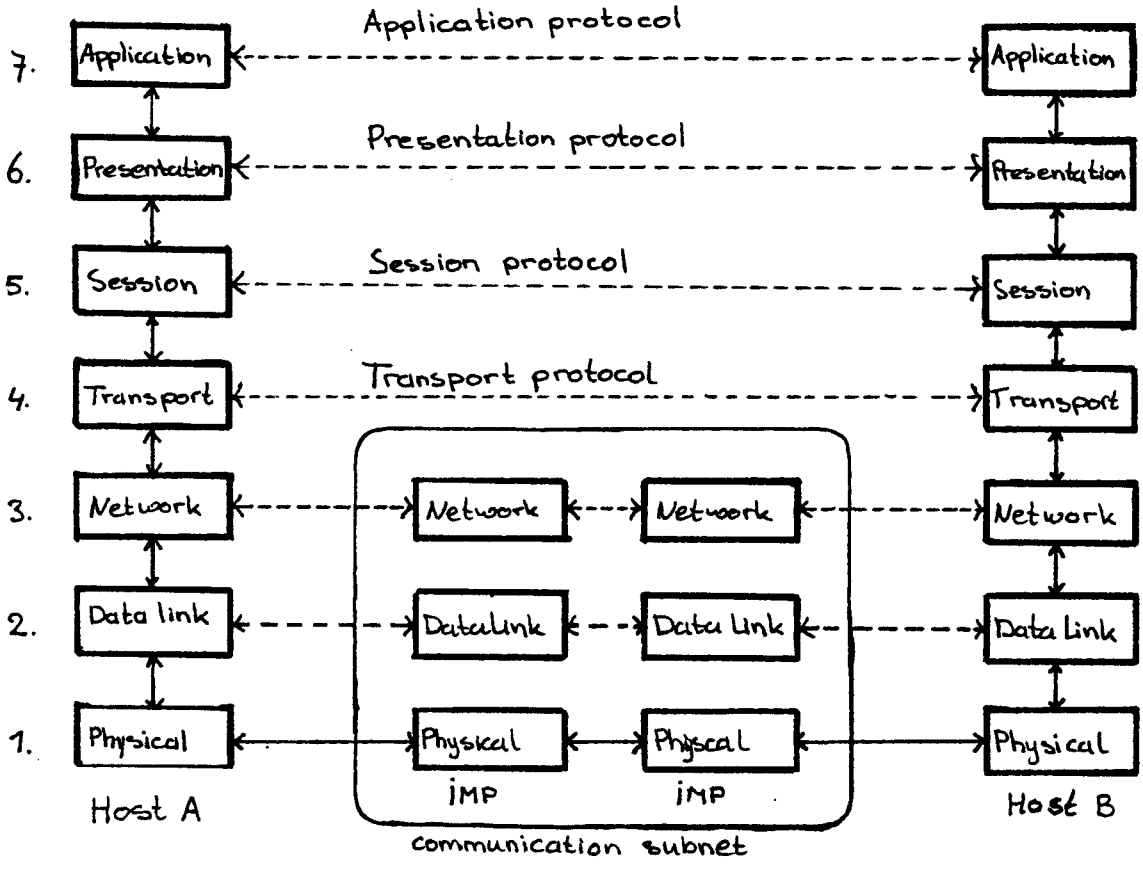


fig.4 het ISO netwerk model.

(1) Het onderste level is het Physical Level. Alleen hier vinden de echte fysische transporten van de bits plaats. Op alle hogere levels is er sprake van virtuele transporten.

(2) Het Data Link Level biedt de hogere levels een connectie zonder transmissiefouten. De bits worden op dit level in frames gegroepeerd. Voorts zorgt dit level er voor dat een ontvangende host niet sneller frames aangeboden krijgt dan dat hij ze kan verwerken. Indien een snelle zender een wat tragere ontvanger overspoelt met frames, kan dat tot deadlock leiden. Het Data Link Level is net als het Physical Level geimplementeerd in de hardware van de hosts.

(3) Het Netwerk Level vormt het interface tussen de hosts en het subnet. De transporten geschieden in packets. Dit level verzorgt de taken van de subnet manager. Het Netwerk Level garandeert de hogere levels dat de packets na eindige tijd hun bestemming, in de juiste volgorde, bereiken. Daartoe moet de subnet manager deadlock kunnen voorkomen, of signaleren en op kunnen lossen.

Merlin en Schweitzer ([15] en[16]) onderscheiden in dit verband de volgende typen deadlock naar de reden van hun ontstaan:

A. Store-and-forward deadlock.
In een verzameling volle buffers wachten een aantal packets op transport. Zij kunnen echter alleen maar naar buffers uit die verzameling getransporteerd worden. De methode die Merlin en Schweitzer geven om dit type deadlock te voorkomen is gebaseerd op het voorkomen van cycles in een gerichte graph (de zgn. buffergraph, zie hoofdstuk IV).

B. Reassembly deadlock
Wanneer grote messages in meerdere packets verdeeld zijn, kunnen die packets in een willekeurige volgorde op hun bestemming aankomen. De ontvangende IMP moet deze packets in zijn buffers opslaan en zelf in de juiste volgorde zetten. Als de bufferruimte van de IMP onvoldoende is, kan er deadlock ontstaan.

C. Pacing deadlock.
Dit type deadlock kan ontstaan doordat devices slechts een beperkte capaciteit hebben. Wanneer een host zijn buffers vol heeft met packets voor zijn devices, is hij niet meer in staat om control messages van die devices te ontvangen. Daardoor krijgt een host ook geen message waarin een device om het volgende packet vraagt, en weet dus niet wanneer hij packets door moet sturen naar een device.
Dit type deadlock is eenvoudig te voorkomen door er voor te zorgen dat elke host minstens een buffer heeft waarin geen andere dan controlpackets van de devices mogen zitten.

D. Copy-release deadlock.
De zendende IMP zal veelal een copie van het packet in zijn buffers bewaren totdat hij een control message krijgt dat het packet zijn bestemming bereikt heeft. Wanneer de buffers van die IMP allemaal bezet zijn kan hij deze control message niet ontvangen, en dus ook de copie niet opruimen.

De subnet manager behoeft zich niet bezig te houden met de feitelijke transporten. Daarvoor zorgen de protocollen van de lagere levels van de netwerk architectuur.
De overige levels die in het ISO model onderscheiden worden zijn dan nog de volgende:
(4) Het Transport Level verzorgt de (virtuele) transporten van messages tussen de hosts (die niet fysisch verbonden behoeven te zijn).
(5) Het Session Level vormt het interface tussen de gebruikers en het netwerk. Vaak is dit level geheel afwezig, of opgegaan in het Transport Level.
(6) Het Presentation Level voert voor de gebruikers veelgevraagde functies uit, zoals bijvoorbeeld tekstcompressie en character- en fileconversies.
(7) De inhoud van het Application Level wordt geheel door de gebruikers bepaald.

Een van de belangrijkste aspecten van de implementatie van een gedistribueerd NWOS is de interproces communicatie (IPC). Er bestaan twee mechanismen voor IPC: function calls en message passing.

A. Function calls.

Het hele netwerk operating system bestaat uit functies (procedures) waarvan de code over de hosts verdeeld is.
Twee hosts communiceren doordat een functie op de ene machine een functie op een remote host aanroept. De caller wacht tot de functie uitgevoerd is.
Parameter overdracht geschiedt door 'call-by-value' als de func-

ties geen gemeenschappelijke bufferruimte hebben. Indien er gemeenschappelijke buffers zijn, is 'call-by-reference' mogelijk. Er is dan wel een synchronisatie mechanisme nodig (zoals een monitor of semaphores).

Doordat het systeem als een groot programma geschreven is, is het niet erg flexibel. Alle functies zijn in dezelfde taal geschreven. De scope-rules van die taal bepalen welke functies met elkaar kunnen communiceren.

B. Message passing.

Het netwerk operating system bestaat uit een aantal verspreide processen. De code van elk proces staat geheel los van die van de andere processen, en kan zelfs in een andere programmeertaal geschreven zijn. Processen kunnen communiceren door uitwisseling van messages.

Een proces moet kenbaar maken voor wie een message bestemd is. Het systeem beslist dan of de processen met elkaar mogen communiceren. Er is een protocol nodig dat de initialisatie van de verbinding en de sequence-, error- en flow control verzorgt.

Message passing is een erg flexibel IPC mechanisme. Daardoor is het ook moeilijk te implementeren. Een vereiste voor de implementatie is dat in de architectuur van het netwerk al bij voorbaat rekening gehouden moet worden met message-passing.

III. Centrale versus decentrale netwerk operating systems.

De mate van (de)centralisatie van het NWOS wordt bepaalt door het aantal managers dat elke taak uitvoert, en door de hoeveelheid communicatie tussen die managers.

In een volledig centraal NWOS is er een (super)manager die alle taken vervult. Er is dus alleen maar communicatie tussen de locale hosts en deze manager nodig. De manager bevindt zich in zijn geheel in een van de hosts, of in een aparte machine.

Een nadeel van een centraal NWOS is de onbetrouwbaarheid van het systeem. Wanneer de machine waarop de manager draait uitvalt, dan is het hele operating system niet meer te gebruiken.

Een tweede nadeel ontstaat doordat er voor al de informatie die de manager van de hosts krijgt, extra veel informatietransporten nodig zijn naar die ene manager. Hierdoor raken de communicatielijnen in de buurt van de machine waar de manager zich bevindt, extra zwaar belast, waardoor er vertragingen kunnen ontstaan.

Een voordeel van een centraal NWOS is de eenvoud van het systeem; het maakt de implementatie relatief gemakkelijk. De implementatie van een centraal systeem is vergelijkbaar met de implementatie van een multiprocessor operating system.

Een wat minder extreme centralisatie van het NWOS wordt bereikt als er meerdere managers zijn, die elk een deel van de taken uitvoeren, en die op verschillende locale hosts geimplementeerd zijn.

In een geheel gedistribueerd NWOS voert elke locale host geheel zelfstandig de globale taken van het NWOS uit, zelfs zonder communicatie met de andere hosts. Globale beslissingen worden uitsluitend op grond van locaal beschikbare informatie genomen.

Wanneer de locale hosts wel onderling communiceren ten behoeve van het NWOS, geeft dat een extra belasting van het subnet. De globale taken zullen echter wel efficienter uitgevoerd kunnen worden doordat de locale managers over wat meer dan alleen locale informatie

beschikken.

Een mogelijk voordeel van een gedistribueerd NWOS ligt in het feit dan de problemen opgelost worden op de plaats waar ze ontstaan. In ieder geval is een gedistribueerd NWOS meer betrouwbaar dan een centraal systeem. Het uitvallen van een van de hosts heeft voor de werking van het NWOS als geheel geen gevolgen.

De implementatie van een gedistribueerd NWOS is veel moeilijker dan de implementatie van een centraal NWOS.

IV. Onderdelen van een gedistribueerd netwerk operating system.

De toestand van het netwerk kan voortdurend aan veranderingen onderhevig zijn; IMP's en communicatielijnen vallen uit, of komen juist weer in bedrijf, en de verkeersintensiteit wijzigt steeds. Een netwerk operating system dient de orde in deze toestandsveranderingen te bewaren.

IV.1. Resource management, het object model.

Een object (Jones[6]) bestaat uit een verzameling datastructuren en een verzameling operaties die daarop uitgevoerd kunnen worden.

Twee objecten heten van het zelfde type als ze dezelfde verzameling operaties hebben. Een object is een representatie van een resource. Een operating system kan beschreven worden in termen van types, waarbij elk type een bepaalde soort van resources representeert (bv. files, I/O-devices).

Een object wordt geimplementeerd door een server proces. Twee servers die een object van het zelfde type implementeren kunnen dat heel verschillend doen. Deze eigenschap maakt het mogelijk om een NWOS te implementeren met behulp van het object model. De verschillen die in de hard- en software van de locale hosts optreden kunnen door de servers gecamoufleerd worden.

Een proces moet, om op een object een operatie te kunnen uitvoeren, in het bezit zijn van een zogenaamde 'capability'. De taak van het NWOS resource management in een object model bestaat uit het beheer van de capabilities.

IV.2. Database management.

Processen die lees- en/of schrijfopdrachten voor delen van een database bevatten, bezoeken achtereenvolgens de verschillende hosts waar de betreffende delen van de database opgeslagen zijn. Op de host waar zo'n proces op een bepaald moment is, heet het proces actief. Op alle hosts die het proces reeds bezocht heeft heet het proces inactief.

De activiteit van een proces kan alleen stoppen door terminatie van het proces, of door de executie van het proces vroegtijdig af te breken. De terminatie van een proces bestaat uit het stoppen van de executie, en het beschikbaar stellen van alle veranderingen die het proces in de database aangebracht heeft, aan de volgende processen. Als een proces voortijdig afgebroken wordt, stopt de executie, en worden alle door het proces reeds aangebrachte veranderingen in de database op de inactieve hosts ongedaan gemaakt (rollback).

De volgende definities en stellingen zijn ontleend aan Rosenkrantz [18].

Definitie: Een concurrency control heet lineariserend, indien het effect op de database van de executie van een aantal concurrente processen tot terminatie, het zelfde is als wanneer de processen in een of andere sequentiele volgorde tot terminatie geexecuteerd waren.

Stelling: Een lineariserende concurrency control behoudt ten allen tijde de consistentie van de database.

Definitie: Twee verzoeken tot lezen en/of schrijven van een eenheid van de database die door verschillende processen worden gedaan, heten conflicterend, indien een van beide een verzoek tot schrijven is, en de host waar de eenheid zich bevindt nog geen terminatie- of rollback-message van een van beide processen heeft gehad.

Conflicten kunnen veroorzaakt worden door een leesopdracht voor een eenheid waarvoor al een schrijfopdracht gegeven is, en door een schrijfopdracht voor een eenheid waarvoor reeds een lees- of een schrijfopdracht gegeven is. Het optreden van conflicten heeft niets te maken met het al of niet toegewezen zijn van een verzoek tot lezen of schrijven van de database.

Definitie: Een concurrency control algorithme heet strikt, indien een verzoek van een proces tot lezen of schrijven van de database niet toegewezen wordt wanneer het in conflict is met een reeds eerder toegewezen verzoek van een ander proces.

Definitie: De current database is de database zoals hij er uit zou zien wanneer alle nog niet getermineerde processen voortijdig afgebroken zouden worden.

Stelling: Bij een strikt concurrency control is de current database te verkrijgen door linearisatie van de getermineerde processen, geordend naar het tijdstip waarop ze termineerden.

Er bestaan niet strikte concurrency control algorithmen die lineariseerbaar zijn. Hierbij valt de benodigde linearisatie dus niet samen met de terminatie volgorde van de processen.

Definitie: Een proces is 'running for ever' indien elk verzoek (eventueel na een eindige wachttijd) toegewezen wordt, maar het proces niet termineert en ook niet voortijdig afgebroken wordt.

Stelling: Bij een strikt concurrency control is geen enkel proces 'running for ever'.

De aanname dat een database uit een eindig aantal eenheden bestaat is voor deze stelling van wezenlijk belang.

Stelling: Een concurrency control dat een verzoek niet toewijst tenzij er een terminatie- of rollbackmessage ontvangen is voor elk van de processen waarmee het verzoek een conflict veroorzaakte, is een strikt concurrency control.

Het concurrency control moet conflicten zodanig behandelen dat elk proces in eindige tijd termineert.

Definitie: Een superstrikt concurrency control heeft de in de voorgaande stelling genoemde eigenschappen (en is dus strikt). Bovendien

laat het een proces alleen maar wachten op processen waarmee het een conflict veroorzaakte.

Stelling: Een superstrikt concurrency control algorithme dat deadlock uitsluit, en dat processen herstart die in conflict zijn met lager genummerde processen, garandeert dan elk proces termineert.

In dit verband verstaan we onder deadlock de situatie dat er een oneindig grote verzameling processen is die allen op elkaar wachten. Een proces wordt herstart nadat het voortijdig gestopt is. De stelling gaat er van uit dat er een methode is om aan elk proces een uniek positief nummer toe te wijzen (zie bv. Lomet[13]).

Rosenkrantz[18] geeft twee gedistribueerde concurrency control algorithmen. Hij veronderstelt dat elk proces extra informatie met zich meedraagt over de reeds bezochte hosts. Deze informatie wordt gebruikt wanneer het concurrency control rollback of termination messages gaat verzenden.

Wanneer een verzoek tot lezen van of schrijven op de database een conflict veroorzaakt, onderneemt het concurrency control een van de volgende acties.

1. WAIT: Het proces dat het conflict veroorzaakte wacht tot de executie van alle processen waarmee het in conflict is getermineerd of afgebroken zijn.

2. RESTART: Een restart van een proces bestaat uit het afbreken van de executie op de host waar het proces actief is, en het verzenden van rollback messages. Vervolgens wordt het proces opnieuw gestart op zijn initiele host.

Het concurrency control algorithme kan of het proces dat het conflict veroorzaakte, of het proces waarmee het conflict ontstond herstarten. In het eerste geval (DIE) kan het proces direct herstart worden. In het tweede geval (WOUND) zendt het concurrency control een WOUND-message voor het betreffende proces naar alle hosts die het proces reeds bezocht heeft. Wanneer deze message de host bereikt waarop het proces actief is, wordt de executie van het proces afgebroken, en weer gestart op de initiele host, tenzij het proces al getermineerd was.

Bij een RESTART treden nogal wat timing problemen op omdat rollback-, termination- en WOUND messages, en het herstarte proces in een willekeurige volgorde een host kunnen bereiken.

Stelling: Wanneer een concurrency control een proces dat een conflict veroorzaakt of herstart (DIE), of laat wachten op de processen waarmee het in conflict is (WAIT), of deze processen laat verwonden (WOUND), dan is het een strikt concurrency control.

We beschouwen enkele van de mogelijke concurrency controls ([18]). Zij Q een proces dat een conflict veroorzaakt met een reeds eerder gemaakt verzoek van proces P, en N_p en N_q unieke positive getallen (de leeftijden van P en Q). Het zogenaamde WAIT-en-DIE concurrency control behandelt conflicten als volgt: wanneer Q ouder is dan P ($N_q<N_p$) wacht Q op P (WAIT), en anders wordt Q herstart (DIE).

Stelling: WAIT-en-DIE is een strikt concurrency control waarvoor elk proces termineert.

Het WOUND-en-WAIT concurrency control behandelt conflicten als volgt: wanneer Q ouder is dan P, wordt P herstart (WOUND), en anders wacht Q

op P (WAIT).

Stelling: WOUND-en-WAIT is een strikt concurrency control waarvoor elk proces termineert.

In het WAIT-en-DIE systeem wachten oudere processen op jongere. Daardoor worden ze steeds ouder en moeten ze steeds vaker wachten. Een ouder proces krijgt in het WOUND-en-WAIT systeem juist een steeds hogere prioriteit doordat het nooit op jongere processen behoeft te wachten.

Een proces Q dat het conflict veroorzaakte met P, doet dat in het WAIT-en-DIE systeem steeds opnieuw zolang P niet getermineerd is. Het wordt dus mogelijk vaak tevergeefs herstart. In het WOUND-en-WAIT systeem kan echter nooit tweemaal achtereen hetzelfde conflict optreden doordat er steeds een ander proces afgebroken en opnieuw gestart wordt.

IV.3. Subnet management: routing.

We bezien enkele mogelijke routerings algorithmen voor packet-switching netwerken.

A. Een adaptief centraal routerings algorithme.

Ergens in het netwerk bevindt zich het Routing Control Center (RCC). Na een bepaalde tijd stuurt elke IMP informatie over de lengte van de wachtrijen, het gemiddeld delay, de status van de communicatielijnen en het al of niet werken van de IMP's waarmee hij direct verbonden is, naar het RCC. Het RCC bepaalt op grond van deze gegevens nieuwe routerings tabellen, en verspreidt deze tabellen over de IMP's.

Doordat het RCC erg veel informatie heeft over zowel de topologie als de verkeersintensiteit, kan het optimale routes voor de packets bepalen.

Om steeds snel op veranderingen in het netwerk te kunnen reageren moet het RCC erg vaak nieuwe tabellen maken. Dat kost, zeker voor een groot netwerk, veel tijd. Het verspreiden van de nieuwe routerings tabellen geeft een extra belasting op het subnet. Bovendien bereiken de nieuwe tabellen de dicht bij het RCC gelegen IMP's eerder, dan de IMP's op grotere afstand van het RCC. Hierdoor kunnen de tabellen van de IMP's gedurende enige tijd inconsistent zijn.

B. Een adaptief gedistribueerd routerings algorithme.

Een routerings algorithme bekend onder de naam 'Backward learning' (Baran[1]), bepaalt voor elke IMP de tabellen met behulp van informatie die de IMP zelf verzamelt. Elke IMP haalt deze informatie uit de packets die hij ontvangt. Elk packet draagt de naam van de IMP waar hij het netwerk binnen gekomen is, en een teller die aangeeft hoeveel hops het packet gemaakt heeft.

De IMP's hebben in een tabel staan wat volgens hun de beste outputlijn voor elke bestemming is, en hoeveel hops een packet dan moet maken om zijn bestemming te bereiken.

Wanneer een packet van een IMP G in IMP H arriveert, langs lijn k, en H ontdekt dat het packet minder hops gemaakt heeft dan dat hij in zijn tabel heeft staan bij de outputlijn naar G, dan verandert H zijn tabel: k wordt nu de beste outputlijn voor packets die van H naar G zouden moeten.

De IMP's reageren alleen op verbeteringen. Als er lijnen of hosts uitvallen, of als er lijnen overbezet raken, komen de IMP's dat

niet te weten. Daarom moet elke IMP na verloop van tijd al zijn vergaarde kennis weggooien, en zijn tabel opnieuw opbouwen.

Tijdens de opbouwperiode versturen de IMP's de packets langs lijnen waarvan ze niet weten hoe goed ze zijn. Wanneer de tabellen slechts heel zelden vernieuwd worden, dan zal het routerings algorithme traag reageren op veranderingen in de topologie van het netwerk, en de verkeersintensiteit.

C. Een gedistribueerd routerings algorithme met in elke IMP volledige informatie over de topologie van het netwerk.

Het Arpanet gebruikt sinds 1979 een routerings algorithme waarbij de IMP's elkaar op de hoogte houden van veranderingen in de topologie. Wanneer de topologie verandert, passen de buren van de IMP waar deze verandering optrad, hun routerings tabellen aan en geven de wijziging door aan hun buren. Omdat alle IMP's de globale topologie kennen kan bij het versturen van deze control informatie voorkomen worden dat er te veel control packets verzonden worden. Voor een netwerk met n IMP's worden er hoogstens n-1 control packets verzonden.

Een beter algorithme, dat de snelheid waarmee de routerings tabellen bijgewerkt worden, maximaliseert en het aantal control packets minimaliseert, bestaat niet.

D. Een gedistribueerd routerings algorithme met informatie over de buren.

Tajibnapis[21] geeft een routerings algorithme voor een netwerk waarin de IMP's alleen hun buren kennen. De IMP's hebben geen informatie over de globale topologie van het netwerk. In Tajibnapis' algorithme is de routing informatie bij elke IMP opgeslagen in een afstands tabel. Deze tabel bevat de afstand, in hops, naar elke andere IMP voor elke outputlijn. Een routerings tabel bevat voor elke IMP de outputlijn voor de kortste route, en de lengte van die route.

Wanneer er bij een IMP een verandering in een routerings tabel optreedt, stuurt deze IMP een control message naar zijn buren om hen van de verandering op de hoogte te stellen. Deze buren werken hun routerings- en afstandstabellen bij, en geven deze veranderingen door aan hun buren, ook aan de buur die de eerste control message stuurde. Dit gaat net zo lang door totdat er geen veranderingen meer in de tabellen aangebracht worden.

Tajibnapis[21] heeft bewezen dat deze methode in eindige tijd tot convergentie leidt, mits maar een eindig aantal veranderingen in het netwerk plaats vinden. Het algorithme genereert erg veel control packets.

IV.4. Subnet management: voorkomen van store-and-forward deadlock.

Een packet-switching netwerk is te representeren door een gerichte graph G=(V,E), met V de verzameling van IMP's, en E de verzameling communicatielijnen tussen de IMP's. Het netwerk kan de volgende 'moves' maken.

1. generatie van een packet in een IMP,
2. IMP G stuurt een packet naar IMP H: in G komt een buffer vrij,
3. consumptie door de host van een packet dat zijn bestemming bereikt heeft: in de betreffende IMP komt een buffer vrij.

Een controller is een algorithme dat bepaalde moves toestaat of verbiedt. Een controller is deadlock free als het voorkomt dat het

netwerk in een zodanige toestand komt, dat er packets zijn die een door de controller toegestane move nooit kunnen maken.

Een centrale controller moet op de hoogte blijven van de toestand van de IMP's: aantallen volle en vrije buffers, oorsprong en bestemming van de aanwezige packets. Daartoe moeten informatie packets gestuurd worden naar de IMP waar de controller zich bevindt. Deze informatie packets zullen zelf de toestand van het netwerk weer veranderen.

Dit probleem treedt niet op indien er een gedistribueerde controller is. In elke IMP wordt dan uitsluitend met behulp van locaal beschikbare informatie bepaald of een packet gegenereerd geaccepteerd of geconsumeerd kan worden.

Merlin en Schweitzer ([15] en [16]) geven een locale deadlockfree controller voor een store-and-foreward netwerk met een bekende topologie. In elke IMP veronderstellen zij een bekend eindig aantal buffers. Voorts gaan ze er van uit dat het routerings algorithme niet-adaptief en reeds bekend is. Bij de gegeven topologie en de routing tabellen construeren zijn een zogenaamde Buffer Graph (BG) met de volgende eigenschappen:

1. voor elke door het routerings algorithme toegestane route is er tenminste een pad in de BG (guaranteed path),
2. de Buffer Graph heeft geen cycles.

De BG is een gerichte graph waarvan de nodes de buffers in de IMP's voorstellen, en edges alleen mogen bestaan tussen buffers in dezelfde IMP, of tussen buffers in IMP's die via een communicatielijn in het netwerk verbonden zijn.

De controller accepteert een nieuw gegenereerd packet als in de betreffende IMP een buffer op een guaranteed path voor dat packet vrij is. Het packet wordt aan de volgende IMP op zijn route doorgegeven als de volgende buffer op het guaranteed path vrijkomt. Omdat de BG geen cycles heeft, is deze controller deadlock-free.

Door er voor te zorgen dat er voor elke route die het routerings algorithme toestaat, veel paden in de BG zijn, en door path-switching toe te staan, worden de beperkingen die de controller aan de transporten van packets oplegt wat verminderd.

Copy-release deadlock (zie II.4) kan voorkomen worden door eenzelfde controller te gebruiken, maar nu voor een BG die aan meer eisen moet voldoen.

Toueg en Ullman[23] geven vier locale deadlock-free controllers voor een packet-switching netwerk met een vaste, maar niet bekende topologie

Stel elke IMP heeft een bekend eindig aantal buffers b, en de routerings tabellen zijn gegeven, dus onveranderbaar, met een maximale lengte van een route gelijk k.

De toestand van een packet wordt gegeven door:
i, het aantal hops dat het packet sinds het verlaten van zijn oorsprong reeds gemaakt heeft,
j, het aantal hops dat het packet nog maken moet voor het zijn bestemming bereikt.
Er geldt i+j<=k.

De locale toestand van een IMP in een netwerk wordt dan bepaald door de volgende grootheden:
m, het aantal lege buffers,
n, het aantal volle buffers,
$i=(i_0,i_1,\ldots,i_k)$, met i_r het aantal packets in de IMP dat al r hops gemaakt heeft,
$j=(j_0,j_1,\ldots,j_k)$, met j_r het aantal packets in de IMP dat nog r hops maken moet voordat ze hun bestemming bereiken.

De vier deadlock-free controllers zijn voor b>k:

1. (m,j) forward count controller (FC). Een packet kan in een IMP H gegenereerd of geaccepteerd worden als H meer lege buffers heeft dat het aantal hops dat het packet nog maken moet (j<m).
2. (j,j) forward state controller (FS). Een IMP H accepteert een packet dat nog j hops te gaan heeft indien voor alle i<=j geldt: $i<b-\Sigma_{j_r}$.
3. (n,i) backward count controller (BC). Een IMP H accepteert een packet dat al i<=k hops gemaakt heeft alleen indien H hoogstens i volle buffers heeft.
4. (i,i) backward state controller (BS). Een IMP H accepteert een packet dat al i<=k hops gemaakt heeft indien voor alle j>=i geldt: $j>=\Sigma_{i_r}$.

Stelling (Toueg en Ullman[23]): voor b<=k zijn er geen uniforme (b,k) deadlock-free controllers.

Definitie: $S_2 \subseteq S_1$ als elke move die door S_2 toegestaan wordt ook door S_1 wordt toegestaan.

Is $S_2 \subset S_1$ dan is S_1 beter dan S_2 , want S_1 legt minder restricties op.

Stelling: $FC \subset FS$, $BC \subset BS$, $BC \subset FC$, $BS \subset FS$.

Een onmiddellijk gevolg is dat van de vier genoemde controllers de FS controller de beste is.

Toueg[24] definieert voorts het verschijnsel live-lock als de situatie waarin, door oneerlijke scheduling, packets hun bestemming nooit bereiken. Hij geeft een controller die zowel livelock-free als deadlock-free is:

5. De forward-state-elapsed-time controller (FSET). Deze controller accepteert packets die ook door de FS controller geaccepteerd zouden worden, en waarvoor bovendien geldt dat er geen wachtende packets zijn met een grotere 'elapsed-time' die ook door een FS controller geaccepteerd zouden worden. Wachtende packets zijn packets die eerder door de IMP geweigerd zijn.

V. Aspecten van implementatie van een networkwide operating system.

Het NWOS zal over het algemeen geimplementeerd worden als een extra laag, bovenop de locale operating systems van de hosts. Hierdoor blijft alle, op de hosts reeds bestaande, software onaangetast.

Bij locale netwerken (onderlinge afstand tussen de hosts tot ongeveer 1 km.) is het NWOS soms ingebed in het systeem van elk van de host. Alle hosts hebben dan ook hetzelfde operating system (zie bv. Donnelly[3]).

V.1. System calls, interpreter.

Een heel eenvoudige uitbreiding van het locale systeem zodat access tot remote resources mogelijk is, wordt verkregen door het toevoegen van een system call voor het verzenden en het ontvangen van data. Eigenlijk wordt het netwerk dan beschouwd als een extra

resource bij de locale host.

In een dergelijk NWOS vereist een verzoek om access tot een remote resource een andere system call, dan een verzoek om access tot een locale resource.

Dit kan ondervangen worden door een interpreter alle system calls op te laten vangen, en op grond van de inhoud te bepalen of het om een locale of een remote call gaat. De interpreter voert dan de gewenste call uit. Beide access methoden moeten nu dus bekend zijn bij de interpreter.

V.2. Agents.

Een andere manier om een NWOS te implementeren is door gebruik te maken van agents (fig.5).

Elk gebruikersproces krijgt een agent toegewezen. De agent is een proces dat de gebruiker een uniform interface met alle resources van het netwerk biedt. Daartoe voert de agent voor de gebruiker de verschillende access procedures uit die nodig zijn om de beschikking over de resources van het netwerk te krijgen.

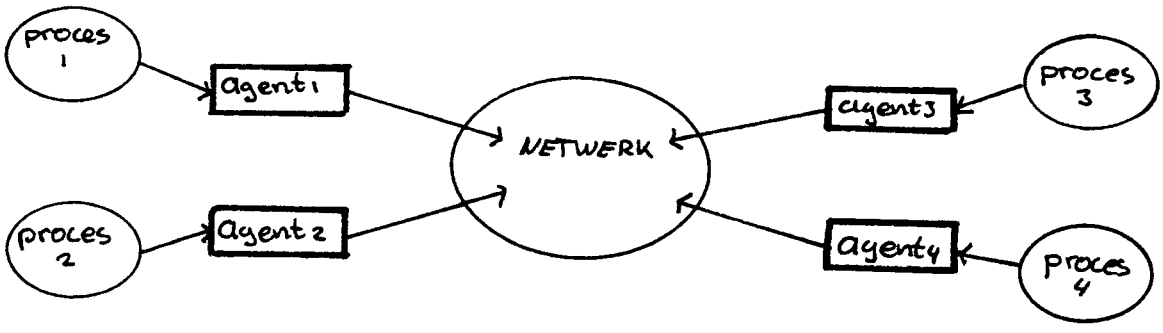


fig. 5 Agents

De agent kan op een van de hosts of op een aparte Network Access Machine geexecuteerd worden.

Een heel eenvoudige agent werkt eigenlijk als een command-processor. Hij vertaalt de commando's van de gebruiker, gegeven in de NCL, in de commando taal van de host waarvoor ze bestemd zijn. Voordat de vertaalde commando's voor executie naar de host verstuurd worden, moet de agent er zeker van zijn, dat alle benodigde files beschikbaar zijn op de host. Eventueel noodzakelijke filetransporten moeten dus al voor de executie van de commando's verzorgd worden. Alle benodigde files moeten dus ook vooraf bekend zijn.

Een meer geavanceerde agent is in staat om tijdens de executie van een gebruikers proces de remote system calls op te vangen, en de gewenste access procedures uit te voeren. Het is dan niet meer nodig om al voor de executie van een proces te weten welke files (resources) er nodig zullen zijn.

V.3. Arpanet.

National Software Works (NSW) is het NWOS van het Arpanet (fig.6). Elk gebruikers proces krijgt een zogenaamd 'front-end' toegewezen. Dat is een proces dat het interface tussen het gebruikers proces en NSW vormt.

Een proces kan alleen gebruik maken van 'tools'; programma's zo-

als compilers, loaders, textprocessors.

De 'worksmanager' is een proces dat een database beheert waarin informatie over accounting, resources, files en tools is opgeslagen. Van deze database kunnen meerdere copien bestaan. Ook kunnen er meerdere worksmanagers zijn.

Wanneer een proces om een tool vraagt, stuurt het front-end een message naar de worksmanager. Deze zoekt naar locaties waar die tool beschikbaar is, en kiest er daar een van uit. Op die host creeert de worksmanager een 'foreman' die de tool tijdens de executie gaat beheren. De foreman vangt alle system calls die de tool maakt op, en voert ze uit.

Voorts is er op elke host een 'file package' aanwezig. Wanneer een tool over een file wil beschikken, stuurt de foreman dat verzoek door naar de worksmanager. Deze kijkt in zijn database waar de gevraagde file aanwezig is, en meldt dat aan de file package van de host waar de tool zich bevindt. Deze file package regelt vervolgens samen met de file package van de remote host, het transport.

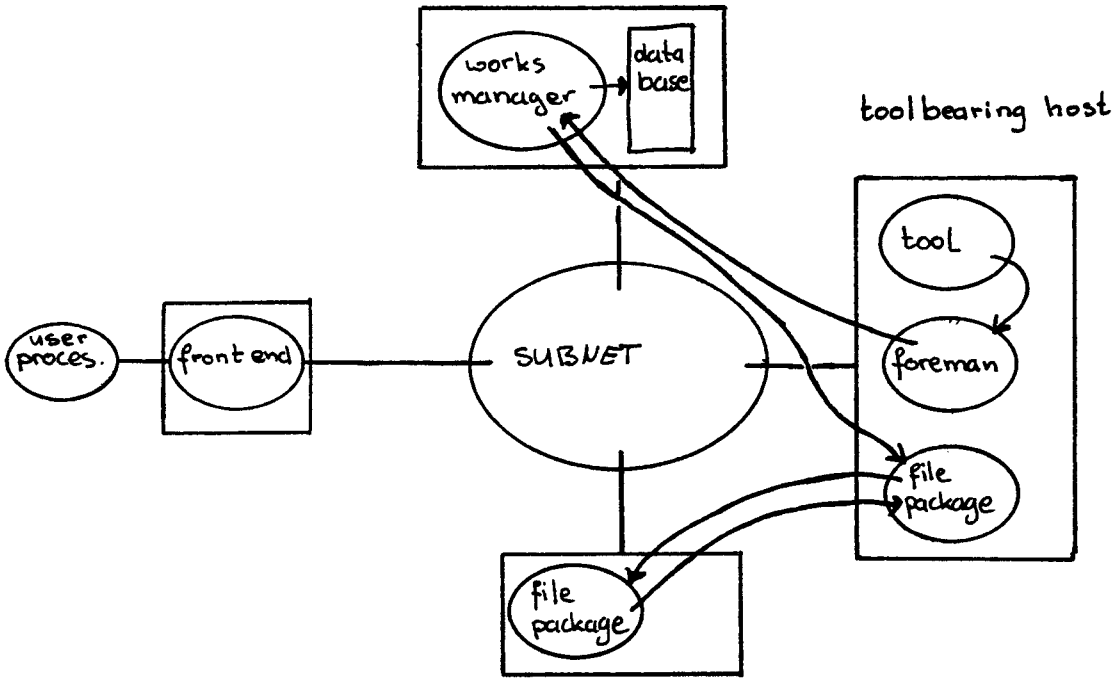


fig. 6. file access in NSW.

De toewijzing van de tools geschiedt dus centraal door de worksmanager. Het beheer van eenmaal toegewezen tools wordt locaal verzorgd door een foreman. Ook wanneer een verzoek tot het openen van een file door de worksmanager is toegewezen, wordt het beheer verder locaal geregeld.

In de loop der tijd hebben de gebruikers van het Arpanet de beschikking over steeds meer tools gekregen. De tools maken uitsluitend NSW calls. Naarmate er meer tools beschikbaar komen zullen gebruikers dus steeds minder locale system calls gaan maken.

Gedetailleerde informatie over diverse netwerken is te vinden in o.a. [22] en [27].

VI. Conclusies.

De organisatie van network operating systems is nog lang niet zo gedetailleerd en systematisch onderzocht als de organisatie van individuele computer operating systems. Een geheel nieuwe ingredient is dat de onderdelen van het operating system gedistribueerd zijn over de hosts en communicatie tussen die onderdelen dient plaats te vinden over het gehele netwerk.

Gedistribueerde netwerk operating systems vereisen gedistribueerde algorithmen voor het beheer van de resources en het subnet. Deze algorithmen zouden, met alleen maar locale informatie ter beschikking, hetzelfde effect moeten hebben als de klassieke centrale algorithmen.

Er wordt thans al wel enig onderzoek verricht aan diverse soorten van gedistribueerde algorithmen voor network operating systems, zoals de gedistribueerde routerings algorithmen (zie bv. Toueg[25] en Jaffe[5]). Ook zijn er inmiddels enkele gedistribueerde concurrency control algorithmen bekend (bv. Rosenkrantz[18]).

Andere problemen dienen zich echter ook aan. Om alle mogelijkheden van het netwerk goed te benutten, moet er in het netwerk bijvoorbeeld aan loadsharing worden gedaan. Er zijn goede algorithmen nodig die de processorallocatie verzorgen.

Veel problemen op dit terrein zijn nog slechts onbevredigend of nog helemaal niet opgelost en vragen om een nauwkeurige studie.

Referenties.

Sommige referenties zijn niet expliciet in de tekst vermeld.

[1] Baran, P., On distributed communication networks, IEEE Trans. Commun. Syst., vol CS-12 (1964) 1-9.

[2] Chorafas, D.N., Computer networks for distributed information systems, Petrocelli, New York, 1980.

[3] Donnelley, J., Components of a network operating system, Computer Networks 3 (1979) 389-399.

[4] Garey, M.R., en D.S. Johnson, Computers and Intractability, Freeman, San Francisco, 1979.

[5] Jaffe, J.M., en F.H. Moss, A responsive distributed routing algorithm for computer networks, Research Report RC8479, IBM TJ Watson Research Center, Yorktown Heights, 1980.

[6] Jones, A.K., The object model: a conceptual tool for structuring software, in: Springer lecture notes in Comp. Sc. No.60, 1978, 7-16.

[7] Kent, S.A., A programmable network virtual machine, Computer Networks 4 (1980) 125-137.

[8] Kimbleton, S.R., en G.M. Schneider, Computer communication networks: Approaches, objectives, and performance considerations, Computing Surveys 7 (1975) 129-173.

[9] Kung, H.T.,en J.T. Robinson, On optimistic methods for concurrency control, ACM Trans. on Database Systems 6 (1981) 213-226.

[10] Lageweg, B.J., et.al., Een geautomatiseerde complexiteits-classificatie van combinatorische problemen, rapport, Mathematisch Centrum Amsterdam, 1981.

[11] Lenstra, J.K., Sequencing by enumerative methods, Proefschrift, Universiteit van Amsterdam, 1976.

[12] Lomet, D.B., Coping with deadlock in distributed systems, Research Report RC7460, IBM TJ Watson Research Center, Yorktown Heights, 1978.

[13] Lomet, D,B., The ordering of activities in distributed systems, Research Report RC8450, IBM TJ Watson Research Center, Yorktown Heights, 1980.

[14] Mahmoud, S.,en J.S. Riordon, Optimal allocation of resources in distributed information networks, ACM Trans. on Database Systems 1 (1976) 66-78.

[15] Merlin, P.M., en P.J. Schweitzer, Deadlock avoidance in store-and-forward networks I: Store-and-forward deadlock, IEEE Trans. on Commun. 28 (1980) 345-354.

[16] Merlin, P.M., en P.J. Schweitzer, Deadlock avoidance in store-and-forward networks II: Other deadlock types, IEEE Trans. on Commun. 28 (1980) 355-360.

[17] Rinnooy Kan, A.H.G., Machine scheduling problems, Proefschrift, Universiteit van Amsterdam, 1976.

[18] Rosenkrantz, D.J. et.al., System level concurrency control for distributed database systems, ACM Trans. on Database Systems 3 (1978) 178-198.

[19] Schwartz, M., Routing and flow control in data networks, Research Report RC8353, IBM TJ Watson Research Center, Yorktown Heights, 1980.

[20] Srinivasan, B., Parallel searching in distributed databases, Computer Networks 4 (1980) 157-166.

[21] Tajibnapis, W.D., A correcness proof of a topology information maintenance protocol for distributed computer networks, Commun. ACM 20 (1977) 477-485.

[22] Tanenbaum, A.S., Computer networks, Prentice Hall, Englewood Cliffs, 1981.

[23] Toueg, S., en J.D. Ullman, Deadlock-free packet switching networks, Procedings 11th ACM symp. on the theory of computing, Atlanta, 1979, 89-98.

[24] Toueg, S., Deadlock- and livelock-free packet switching networks, Procedings 12th ACM symp. on the theory of computing, Los Angelos California, 1980, 94-99.

[25] Toueg, S., A minimum-hop path failsafe and loop-free distributed algorithm, Research Report RC8530, IBM TJ Watson Research Center, Yorktown Heights, 1980.

[26] Watson, R.W., en J.G. Fletcher, An architecture for support of network operating system services, Computer Networks 4 (1980) 33-49.

[27] Springer Lecture notes in Comp.Sc. No.105, Distributed systems: Architecture and implementation, Springer Verlag Heidelberg, 1981.