



**GEKODEERDE X-STRAALBEELDTRANSMISSIE OP 'n SELLULÊRE
DATANETWERKKOPPELING VIR GEBRUIK IN MOBIELE MEDIESE
TOEPASSINGS**

deur

PIETER STEFANUS VELDTSMAN

Verhandeling voorgelê ter voldoening aan die vereistes vir die:

MAGISTER TECHNOLOGIAE:

Ingenieurswese: Elektries

In die Fakulteit Ingenieurswese aan die Technikon Vrystaat

Datum van inhandiging: JULIE 1996

Studieleier: Prof. G.J. Prinsloo

BEDANKINGS

Ek wil graag die volgende persone en instansies bedank vir hul hulp en ondersteuning ter voltooiing van die projek:

Mnr. M.S. Van Rensburg van The Talk Shop, Bloemfontein, vir die voorsiening van die sellulêre telefoon en simkaarte vir die doen van die toetse.

Mnr. J. Bezuidenhout van Transtel, Bloemfontein, vir die voorsiening van toerusting.

Die studieleier, prof. G.J. Prinsloo vir sy hulp en bystand tydens die studie en voltooiing van die projek.

Technikon Vrystaat vir die geleentheid om die kwalifikasie aan die instansie te kan verwerf.

Technikon Vrystaat Bronnesentrum vir daarstelling van geriewe en inligting.

My ouers vir al die geleenthede en ook ondersteuning wat hulle my gebied het om die kwalifikasie te kan verwerf.

My verloofde, Louisa, vir haar bystand en ondersteuning gedurende hierdie tyd.

Alle persone en instansies wat dalk nie hier genoem is nie, maar bereid was om inligting te voorsien.

UITTREKSEL

Daar is 'n behoefte om X-straalbeelde vanaf 'n mobiele mediese eenheid na 'n sentrale rekenaar te stuur. Hierdie versending van die data moet outomaties geskied en dus vinnige, effektiewe oordrag van X-straalbeelde en ander mediese data vanaf 'n mediese voertuig of buitepos moontlik maak. Die stelsel moet eenvoudig wees om te gebruik en die oordragmetode moet toelaat dat data op die sentrale rekenaar gebruik kan word. Dit moet dus 'n netwerkverbinding wees.

Sellulêre tegnologie kan gebruik word om sodanige datakanaal te skep. Alhoewel die sellulêre spektrum primêr vir telefoongesprekke gebruik word, is dataoordrag (rekenaar lêers) ook moontlik. Sellulêre operateurs voorspel dan ook dat die oordrag van data 'n al hoe groter deel van die spektrum sal gebruik. Indien 'n netwerkverbinding deur middel van 'n sellulêre telefoon en modem gemaak sou word, is dit moontlik vir die mobiele eenheid om toegang te hê tot data op die sentrale rekenaar.

Die X-straalbeelde sal gekodeer word om die transmissietyd te verkort. X-straalbeelde kan met behulp van 'n sellulêre- of ander telefoonkoppelvlak en 'n modem gestuur word deur 'n TCP/IP koppeling tussen twee rekenaars op die RS232 poorte, daar te stel. Die oordrag van gekodeerde beelde sal deur die gebruikmaking van die netwerkkoppeling betroubaar en aanpasbaar wees.

Die studie wat gemaak is, dui ook die verskillende redes aan waarom besluit is om TCP/IP as netwerkprotokol te gebruik. Die TCP/IP Internet Protokol het die standaard geword vir die onderlinge verbinding van rekenaarstelsels. Die TCP/IP Internet Protokol word meer as enige ander protokol in stelsels regoor die wêreld

gebruik. Dit verseker dus 'n groot mate van versoenbaarheid tussen netwerkstelsels. Navorsings- en opvoedkundige instansies gebruik TCP/IP as die primêre grondslag vir datakommunikasie. Daar word na TCP/IP verwys as die Internet Protokol reeks, aangesien dit die vermoë beskik om stelsels, wat aan verskillende tipes netwerke gekoppel is, te laat kommunikeer.

SUMMARY

There is a need for sending X-ray images from a mobile medical unit to a central computer. This transfer must be automatic and fast. Effective transfer of the X-ray images and other medical data from the medical vehicle or outpost must be possible. The system must be simple to use and the transfer method must allow the use of data on the central computer. Therefore a network connection is used.

Cellular technology can be used to create such a data channel. Although the cellular spectrum is primarily used for telephone conversations, data transfer (computer files) is also possible. Cellular operators predict that data transfer will increasingly use more of the cellular spectrum. If a network link is made by means of a cellular telephone and modem, it is possible for the mobile unit to gain access to data on the central computer.

The X-ray images are compressed to reduce transmission time. X-ray images can be sent with the aid of a cellular- or other telephone interface with a modem, by establishing a TCP/IP link between two computers on their RS232 ports. By using a network link, the transfer of the compressed images is reliable and adaptable.

The study that was made shows the reasons for choosing TCP/IP as a network protocol. The TCP/IP Internet Protocol became the standard for the interconnection of computer systems. The TCP/IP Internet Protocol is more widely used throughout the world than any other protocol. This ensures a great deal of compatibility between network systems. Research- and educational institutions use TCP/IP as the primary basis for data communications. TCP/IP is referred to as

the Internet Protocol series because it has the ability to ensure that systems that are connected to different types of networks can communicate.

INHOUD

Bladsy

HOOFSTUK 1

INLEIDING

1.1	Oorsig	1
1.2	Probleemstelling en oplossing	2
1.3	Doel van die studie	2
1.4	Hipotese	3
1.5	Belangrikheid van die projek	3
1.6	Metode van navorsing	4
1.6.1	Ontwikkeling van die netwerkontvanger sagteware	4
1.6.2	Ontwikkeling van sagteware vir die netwerksender	5
1.6.3	Evaluering van die stelsel deur middel van 'n nulmodemkabel	5
1.6.4	Ontwikkeling van die modemkoppelvlak sagteware	5
1.6.5	Implementering van sellulêre koppeling	6
1.6.6	Evaluering van die totale stelsel	6
1.7	Uniekheid van projek	7
1.8	Opsomming	7

HOOFSTUK 2

KOMMUNIKASIEMEDIUM

2.1	Kriteria vir die vergelyking van die verskeie beskikbare mediums	8
-----	--	---

2.2	Sellulêre tegnologie	9
2.2.1	Werking van die sellulêre telefoonnetwerk	9
2.2.2	Verspreiding van selle	10
2.2.3	Mobiele senders	14
2.2.4	Modulasietegnieke	15
2.2.5	Kodering vir spraak in sellulêre kommunikasie	17
2.2.6	Kodering vir dataoordrag in sellulêre kommunikasie	19
2.2.7	Enkripsie tegnieke in sellulêre stelsels	21
2.2.8	Voordele van 'n sellulêre stelsel	22
2.2.9	Nadele van 'n sellulêre stelsel	22
2.3	Satellietkoppelings	23
2.3.1	Werking van VSAT-stelsels	24
2.3.2	Frekwensiebande wat gebruik word vir satellietkommunikasie	26
2.3.3	Voordele van VSAT-stelsels	27
2.3.4	Nadele van VSAT-stelsels	27
2.4	Radioverbindings	28
2.4.1	Werking van radiotoerusting en netwerke	28
2.4.2	Modulasietegnieke vir pakketradio	29
2.4.3	Voordele van radiokommunikasie	29
2.4.4	Nadele van radiokommunikasie	30
2.5	Keuse van die kommunikasiemedium	30

2.6	Opsomming	31
-----	-----------	----

HOOFSTUK 3

REKENAARNETWERKE

3.1	Rekenaarnetwerke	33
3.2	Netwerkprotokolle	35
3.2.1	Die funksie van 'n netwerkprotokol	35
3.2.2	TCP/IP as netwerkprotokol	35
3.2.3	Die plasing van TCP/IP in die netwerkhiërargie	37
3.2.4	Beskrywing van TCP/IP protokolle	38
3.2.5	Die TCP vlak	43
3.2.6	Die IP vlak	47
3.2.7	Koppelpunte bekend aan TCP netwerk	49
3.2.8	Roetebepaling van die TCP netwerk	51
3.2.9	Datagramfragmentasie en hersamestelling	52
3.3	Ander protokolle as TCP: UDP en ICMP	53
3.4	NetBIOS as netwerkprotokol	54
3.4.1	NetBIOS in die netwerkhiërargie	54
3.5	Opsomming	55

HOOFSTUK 4

SAGTEWARE ONTWIKKELING

4.1	Algemene beskrywing van die sagteware	56
4.1.1	Ontwikkeling van die netwerkontvanger sagteware	56
4.1.2	Ontwikkeling van sagteware vir die netwerksender	64
4.1.3	Ontwikkeling van die modemkoppelvlak sagteware	74
4.2	Opsomming	78

HOOFSTUK 5

EVALUERING VAN STELSEL

5.1	Eksperiment 1: Aantekening	81
5.1.1	Doel	81
5.1.2	Metode	81
5.1.3	Resultate	81
5.1.4	Gevolgtrekking	82
5.2	Eksperiment 2: Dataoordrag en modemsakeling	83
5.2.1	Doel	83
5.2.2	Metode	83
5.2.3	Resultate	83
5.2.4	Gevolgtrekking	84
5.3	Eksperiment 3: Sellulêre koppeling	85
5.3.1	Doel	85

5.3.2 Metode	85
5.3.3 Resultate	85
5.3.4 Gevolgtrekking	86
5.4 Gevolgtrekking	87
HOOFSUK 6	
SAMEVATTING	90
BYLAAG 1 - Sellulêre dekkingskaart	92
BYLAAG 2 - HOST.C	93
BYLAAG 3 - HOSTMOD.CPP	115
BYLAAG 4 - SERVER.C	119
BYLAAG 5 - SERVMOD.CPP	131
BYLAAG 6 - COMPARE.C	135
BYLAAG 7 - KONFIGURASIELêER	137
BYLAAG 8 - API.H	138
BYLAAG 9 - SCLASS.H	142
LITERATUURLYS	147

Lys van tabelle en figure

Figuur 2.1a	'n Sellulêre stelsel	10
Figuur 2.1b	'n Sellulêre stelsel	10
Figuur 2.2	Basisstasies in die middel van elke sel, met frekwensiepare f_1 , f_2 en f_3	11
Figuur 2.3	Direksionele antennes op die nodus van die selle	11
Figuur 2.4	Ideale situasie met sewe pare frekwensies	12
Figuur 2.5	Vier frekwensiepare in gebruik	12
Figuur 2.6	Groter patroon gevorm deur vier pare frekwensies	12
Figuur 2.7	Agt tydgleuwe per TDMA raam	15
Figuur 2.8	Blok data van monsters geneem	17
Figuur 2.9	Stappe wat gevolg word om 'n nuwe blok data te vorm	17
Figuur 2.10	Hersamestelling van nuwe blok data	18
Figuur 2.11	Normale sarsie	19
Figuur 2.12	Vier blokke data van sestig bisse elk	20
Figuur 2.13	Sestig bisse elke 10 millisekonde	20
Figuur 2.14	Invoeging van stertbisse	21
Figuur 2.15	'n VSAT stelsel	24
Figuur 3.1	OSI Netwerkhiërargie verwysingsmodel	38
Figuur 3.2	TCP/IP in die OSI verwysingsmodel	38
Figuur 3.3	Voorstelling van 'n datastroom	44
Figuur 3.4	Opbreking van datastroom in datagramme	44
Figuur 3.5	TCP kopetiket	46
Figuur 3.6	Samevoeging van TCP kopetiket vooraan elke datagram	46
Figuur 3.7	IP kopetiket	48

Figuur 3.8	Samevoeging van IP en TCP kopetikette, sowel as datagramme	49
Figuur 3.9	NetBIOS in die netwerkhiërargie	54
Figuur 4.1	Vloeydiagram van ontvanger sagteware	63
Figuur 4.2	Vloeydiagram van sender sagteware	73
Figuur 4.3	Skakelproses gedoen deur modem	77
Tabel 2.1	Kommunikasiesatelliet frekwensietoekenings (MHz)	26
Tabel 2.2	Vergelyking van verskillende mediums volgens kriteria	31
Tabel 3.1	Klasse van die IP adres	41
Tabel 3.2	Welbekende koppelpunte in TCP	50
Tabel 4.1	Kopetiket ingevoeg om datavloei te vergemaklik	65
Tabel 4.2	UART registers en adresse	76
Tabel 5.1	Gemiddelde tye, Standaard afwyking, en variansie van eksperiment 1	82
Tabel 5.2	Gemiddelde tye, Standaard afwyking, en variansie van eksperiment 2	84
Tabel 5.3	Gemiddelde tye, Standaard afwyking, en variansie van eksperiment 3	86
Tabel 5.4	Eksperiment resultate	87

Lys van Akronieme

BSB	Basisstasie Beheerder
DNS	"Domain Name Service"
FTP	"File Transfer Protocol"
GMSK	"Gaussian Minimum Shift Keying"
GSM	"Group Spècial Mobile" of "Global System for Mobile communications"
ICMP	"Internet Control Message Protocol"
IDU	"Indoor Digital Processing Unit"
IP	Internet Protokol
ISO	"International Standards Organization"
LAN	Lokale Area Netwerk
NetBIOS	"Network Basic Input/Output System"
OSI	"Open Systems Interconnection"
ODU	"Outdoor RF unit"
PSTN	Publieke skakeltelefoon netwerk
RBS	Radio Basisstasie
SLIP	"Serial Line Interface Protocol"
TCB	"Transmission Control Block"
TCP	"Transmission Control Protocol"
TDMA	"Time Division Multiple Access"
UDP	"User Datagram Protocol"
VSAT	"Very Small Aperture Terminal"
WAN	Wye Areanetwerk

HOOFSTUK 1

INLEIDING

1.1 Oorsig

'n Meer effektiewe gesondheidsdiens, soos die verkryging van 'n mediese agtergrond van 'n pasiënt vanaf 'n sentrale databasis, kan verskaf word deur van datakommunikasie gebruik te maak om afgeleë gebiede met inligting te bedien.

Hierdie ondersoek is gerig op die verskaffing van 'n meer effektiewe gesondheidsdiens deur die voorsiening van 'n rekenaar databasisverbinding deur 'n reeks kommunikasiekanale. 'n Algemene probleem van bestaande gesondheidsdienste is dat 'n pasiënt dikwels van een distrik na 'n ander verhuis en daar dan geen mediese rekord van hierdie pasiënt by die nuwe distrik se klinieke of hospitale bestaan nie. Daar is dus geen mediese agtergrond, soos byvoorbeeld vorige en huidige behandelings van die pasiënt, vir gebruik deur die mediese personeel nie.

Die Vrystaat provinsie van Suid-Afrika beskik tans oor 160 Provinsiale Owerhede Klinieke en 20 vaste- en 128 mobiele PGS klinieke [34]. Die meeste van die vaste klinieke en hospitale beskik oor kommunikasie deur middel van telefone, hetsy outomaties- of handsentrales, maar vir mobiele klinieke is slegs radios in sekere gevalle beskikbaar.

Vir hierdie mobiele mediese eenhede word dit ook al hoe noodsaakliker om toegang te hê tot 'n deskundige mening by meer komplekse diagnoses soos

byvoorbeeld die interpretasie van X-straal- en sonarbeelde, of uitsonderlike klankdiagnoses met behulp van 'n stetoskoop.

1.2 Probleemstelling en oplossing

Daar is 'n behoefte om X-straalbeelde en ander data vanaf 'n mobiele mediese eenheid na 'n sentrale rekenaar te stuur. Hierdie versending van die data moet outomaties geskied, sonder dat die operateur 'n tegniese agtergrond het, of dat dit nodig sal wees om interaktief die dataoordragproses te beheer.

Mobiele mediese eenhede in afgeleë gebiede het dikwels nie toegang tot 'n kommunikasiekanaal nie. Sellulêre tegnologie bied baie moontlikhede vir hierdie toepassing aangesien die ontwikkeling van sellulêre tegnologie juis gerig is op die verskaffing van mobiele kommunikasie. Aangesien sellulêre dekking in die Vrystaat provinsie van Suid-Afrika op die oomblik nog geensins voldoende is nie, sal alternatiewe fisiese koppelings soos byvoorbeeld satelliet- en radiokoppelings ook ondersoek word. Indien 'n netwerkverbinding deur middel van 'n sellulêre telefoon en 'n modem, of enige ander medium gemaak sou word, is dit moontlik vir die mobiele eenheid om toegang te hê tot data op die netwerkbediener.

1.3 Doel van die studie

Die doel van hierdie projek is die ontwikkeling van die sagteware, wat in C++ geskryf sal word, en die daarstelling van 'n rekenaar netwerkverbinding tussen 'n mobiele mediese eenheid en 'n sentrale mediese sentrum. Sodanige stelsel moet die oordrag van X-straalbeelde, sowel as ander data vanaf die mobiele

eenheid na die sentrale netwerk, moontlik maak. Hoewel daar reeds baie sagteware bestaan wat lêeroordrag moontlik maak, is daar 'n behoefte aan sagteware wat gebruik maak van 'n netwerkkoppeling, wat ook die dataoordrag outomaties sal doen. Dit sal dus nie 'n operateur benodig wat die proses moet monitor of stapsgewys beheer, soos tans die geval met die meeste kommersiële pakette is nie.

1.4 Hipotese

X-straalbeelde kan met behulp van 'n sellulêre- of ander telefoonkoppelvlak en 'n modem gestuur word deur 'n TCP/IP koppeling tussen twee rekenaars op die RS232 koppelvlak daar te stel. Sodanige oordrag van die beelde kan geskied sonder benadeling van die betroubaarheid daarvan.

1.5 Belangrikheid van die projek

Alhoewel Suid-Afrika volgens internasionale standaarde genoeg verpleegsters, dokters en hospitaalbeddens het, is die mediese dienste in Suid-Afrika, as gevolg van die fisiese uitgestrektheid, in 'n sekere sin oneffektief en onvoldoende. Suid-Afrika spandeer huidiglik R550 per kapita per jaar aan gesondheidsdienste, wat tien keer meer is, as die geskatte waarde van die Wêreldbank, om basiese mediese- en kliniekdienste te kan voorsien. Nieteenstaande bogenoemde, is daar nog miljoene mense wat nie toegang tot hierdie dienste het nie. Die situasie is veral swak in die landelike gebiede [35, p. 20].

Mediese dienste word 'n al hoe groter behoefte, maar baie mense is woonagtig in verafgeleë gebiede, waar dikwels geen infrastruktuur bestaan nie. Dit word ook noodsaaklik vir medici om inligting vanaf 'n sentrale databasis en ander mediese inligting na en van 'n sentrale hospitaal te verkry. Om hierdie redes is dit belangrik om 'n betroubare netwerkverbinding tussen 'n mobiele mediese eenheid en 'n sentrale rekenaar te ontwikkel. Mobiele datakommunikasie is egter op hierdie stadium nog baie duur en die mees koste-effektiewe en betroubare opsies moet dus teen mekaar opgeweeg word.

1.6 Metode van navorsing

Die projek is in 6 fases uitgevoer, naamlik:

- Ontwikkeling van die netwerkontvanger sagteware.
- Ontwikkeling van sagteware vir die netwerksender.
- Evaluering van die stelsel deur middel van 'n nulmodemkabel.
- Ontwikkeling van die modemkoppelvlak sagteware.
- Implementering van sellulêre koppeling.
- Evaluering van die totale stelsel.

1.6.1 Ontwikkeling van die netwerkontvanger sagteware

Die sagteware vir die netwerkontvanger is in C++ vir die TCP/IP netwerk protokol geskryf. Die sagteware het dit ten doel om voorbereid te wees op die koppeling van 'n netwerksender en om, onder die beheer van die sender, onmiddelik te begin met ontvangs van datapakette. 'n Wagwoord kan as beskerming van ongemagtigde data gebruik word. Die data word ontvang op die

seriepoort en kan op skyf gestuur word of deurgestuur word na enige netwerk bediener waartoe die betrokke rekenaar toegang het.

1.6.2 Ontwikkeling van sagteware vir die netwerksender

Die ontwikkeling van die sendersagteware, is meer gerig op die vervanging van 'n operateur. Data, soos bv X-straalbeelde, word deur middel van 'n skandeerder ingelees en op 'n sekere indeks gestuur. Die sendersagteware monitor die skyf en indien nuwe data beskikbaar is, word onmiddelik begin met die koppeling na die ontvanger. As die fisiese verbinding bestaan, word die netwerkkoppeling geopen waarna dataoordrag begin.

1.6.3 Evaluering van die stelsel deur middel van 'n nulmodemkabel

Nadat die sagteware tot bogenoemde stadium voltooi is, is die totale stelsel getoets deur gebruik te maak van 'n nulmodemkabel tussen die twee rekenaars se RS232-poorte. Aantekening vanaf die sender na die ontvanger was nou moontlik en lêeroordrag kan outomaties geskied.

1.6.4 Ontwikkeling van die modemkoppelvlak sagteware

Hierdie sagteware hanteer die skakeling sowel as die beskikbaarmaking van die datalyn tussen die twee rekenaars. Die modemkoppelvlak sagteware voer die sekweniële skakelproses stapsgewys uit. Die modem word eers geïnisialiseer waarna die skakelstring, wat die telefoonnommer bevat, na die modem gestuur word. Indien die fisiese verbinding suksesvol gemaak is, word die TCP/IP data

netwerkkoppeling oopgemaak waarna dataoordrag kan begin. Hierdie faset handel oor die hantering van die skakelprotokol op die fisiese telefoon netwerkkoppeling wat nou die fisiese kabel vervang.

1.6.5 Implementering van sellulêre koppeling

Die sellulêre koppeling vereis 'n PCMCIA modem sowel as spesiale kabel wat tussen die modem en die sellulêre telefoon gebruik word. Aangesien 'n PCMCIA modem vereis word, word daar van 'n draagbare ("laptop") rekenaar gebruik gemaak, wat hierdie koppelvlak standaard bevat. Die telefoon sowel as die rekenaar is dus fisies klein en maak beide van battery kragbronne gebruik, wat meer voordelig is, vir die toepassing in mobiele mediese eenhede.

Aangesien die netwerkontvanger gewoonlik by 'n sentrale hospitaal is, waar gewone telefoongeriewe bestaan, is die sagteware vir gewone modems ook ontwikkel. Hierdie sagteware kan dus gebruik word op enige rekenaar wat oor 'n seriepoort beskik.

1.6.6 Evaluering van die totale stelsel

Die evaluering van die stelsel is deurlopend tydens die ontwikkeling van die sagteware gedoen, maar 'n deeglike toetsing van die stelsel is gedoen by die voltooiing van die sagteware. Hierdie toetsing het toetse soos byvoorbeeld die bepaling van fouttempo's sowel as dataverlies ingesluit. 'n Meer volledige uiteensetting van die evaluering word in Hoofstuk 5 gegee.

1.7 Uniekheid van projek

Hoewel daar reeds baie sagteware bestaan wat dataoordrag moontlik maak, is dit belangrik om sagteware te ontwikkel vir die oordrag van mediese inligting, deur gebruik te maak van 'n netwerkkoppeling. Hierdie oordrag moet moontlik wees sonder dat 'n operateur die proses moet monitor soos wat die geval is by meeste kommersiële pakette. Aangesien daar van 'n netwerkkoppeling gebruik gemaak word, sal dit dus ook moontlik wees om ander sagteware, soos databasis sagteware, oor die netwerkkoppeling te gebruik.

1.8 Opsomming

'n Gedetailleerde beskrywing van die verskillende kommunikasiemediums wat oorweeg is vir die projek, kan in Hoofstuk 2 verkry word, terwyl Hoofstuk 3 handel oor die verskillende tipes netwerke sowel as die verskillende netwerkprotokols. Hoofstuk 4 beskryf die sagteware ontwikkelingsproses en Hoofstuk 5 handel oor die evaluering van die stelsel en gee ook die fouttempo's. Hoofstuk 6 gee 'n kort samevatting van die projek.

HOOFSTUK 2

KOMMUNIKASIEMEDIUM

Hierdie hoofstuk beskryf die werking, sowel as die voor- en nadele van die verskillende mediums wat oorweeg is vir die daarstelling van 'n mobiele datakoppeling. Paragraaf 2.1 gee 'n uiteensetting van die kriteria wat gebruik is by die bepaling van die medium wat gebruik moet word. Paragraaf 2.2 handel oor sellulêre tegnologie, terwyl Paragraaf 2.3 satelliettoerusting beskryf. Paragraaf 2.4 behandel radiosender- en ontvangstoestelle. In Paragraaf 2.5 word 'n keuse gemaak en aan die hand van die kriteria gemotiveer.

2.1 Kriteria vir die vergelyking van die verskeie beskikbare mediums

Die verskeie stelsels wat oorweeg is vir die gebruik as kommunikasiemedium van hierdie projek (satelliet, sellulêr en radio), is ten opsigte van die volgende kriteria teenoor mekaar opgeweeg.

- Fisiese grootte van toerusting
- Kragbron vereistes
- Koste (Kapitaal- en bedryfskoste)
- Tyd beskikbaar vir kommunikasie
- Word herleistasies benodig?
- Grootte van dekking
- Antennes

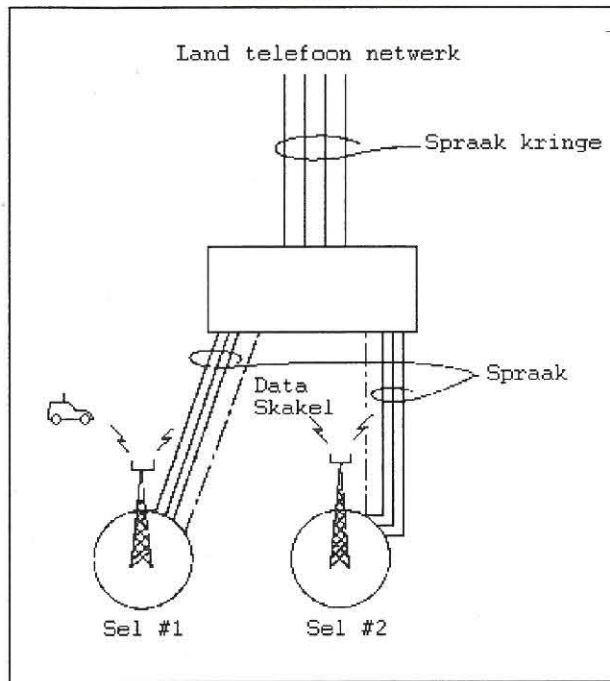
Op grond van bogenoemde is dan besluit op 'n geskikte kommunikasiemedium vir hierdie projek. Die bogenoemde kriteria is gelys op grond van prioriteite, met die hoogste prioriteit boaan die lys.

2.2 Sellulêre tegnologie

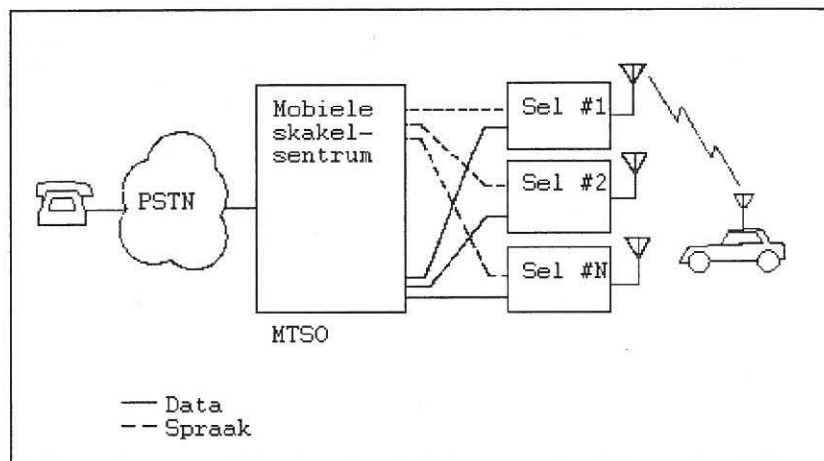
Sellulêre kommunikasie word vandag 'n al hoe gewilder kommunikasiemedium. Die vraag na mobiele kommunikasie kan duidelik uit die gewildheid van hierdie nuwigheid in Suid-Afrika gesien word. Alhoewel dit 'n relatief duur tipe kommunikasie is, word dit reeds baie algemeen deur die publiek gebruik. Daar kan dus ook verwag word dat die dekkingsgebied mettertyd vergroot sal word.

2.2.1 Werking van die sellulêre telefoonnetwerk.

'n Sellulêre telefoonstelsel is basies 'n radiotelefoonstelsel, wat 'n dupleks koppeling met die publieke skakeltelefoonnetwerk (PSTN), bied. Gebruikers van sellulêre telefone kan draagbare telefone of gemonteerde toerusting in 'n voertuig gebruik [17, p. 62]. Figuur 2.1a en 2.1b toon die basiese werking van 'n sellulêre stelsel.



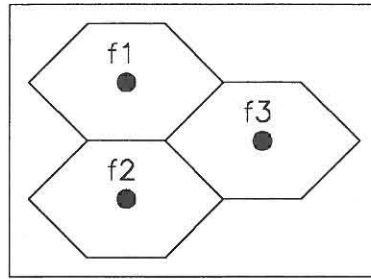
FIGUUR 2.1a 'n Sellulêre stelsel



FIGUUR 2.1b 'n Sellulêre stelsel

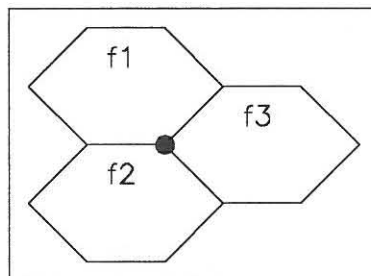
2.2.2 Verspreiding van selle

Soos wat die naam (sellulêre telefoon) aandui, word 'n sekere geografiese gebied verdeel in verskillende selle. Hierdie selle het dan elkeen sy eie Radio Basisstasie (RBS) in die middel (Sien Figuur 2.2) [24, pp. 8, 63].



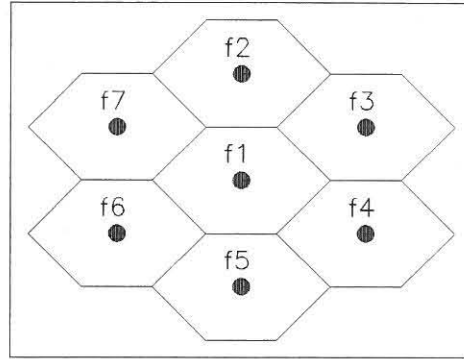
FIGUUR 2.2 Basisstasies in die middel van elke sel, met frekwensiepare f1, f2 en f3

Afhangende van die aantal oproepe wat in 'n sel gemaak sal word, kan 'n sel van 1 tot 50 km in deursnee wees. Een sel kan tot 'n honderd oproepe gelyktydig hanteer, maar wanneer die oproepe te veel word, kan die kapasiteit van die sel maklik verhoog word, deur die sel te verdeel. Dikwels kan een bouterrein drie direksionele antennes hê. Sodoende word drie afsonderlike selle geskep. Die direksionele antennes word dan op die nodus van die drie selle geplaas (Sien Figuur 2.3).



FIGUUR 2.3 Direksionele antennes op die nodus van die selle

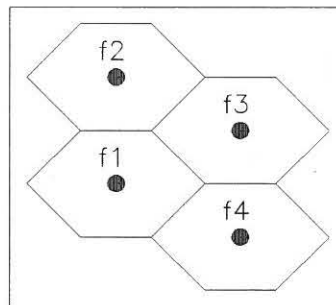
Die ligging van basisstasies is baie belangrik in die algehele selstruktuur van die netwerk. Basisstasies word geselekteer tesame met die toepaslike antenne om die nodige r.f. golfpatroon te verkry. In die onderstaande Figuur 2.4 word die ideale situasie aangedui, waar sewe basisstasies op verskillende pare frekwensies f1 tot f7 gebruik word.



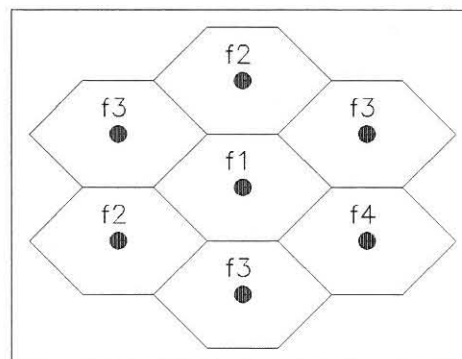
Figuur 2.4 Ideale situasie met sewe pare frekwensies

In werklikheid word slegs vier verskillende pare frekwensies gebruik, om die onderstaande vorm te gee (Sien Figuur 2.5):

Hierdie vier pare word dan so gebruik, om die patroon in Figuur 2.6 te verkry, teneinde te verseker dat dieselfde frekwensie nie in twee aangrensende selle gebruik word nie.



Figuur 2.5 Vier frekwensiepare in gebruik



Figuur 2.6 Groter patroon gevorm deur vier pare frekwensies

Wanneer 'n oproep vanaf 'n sellulêre telefoon gemaak word, word 'n radiosein na die RBS gestuur. 'n Basisstasie Beheerder (BSB) beheer 'n aantal van die radio basisstasies en stuur dan die oproep deur middel van mikrogolfverbinding of optiese vesel na 'n mobiele skakelsentrum. Hierdie skakelsentrum is die brein van die stelsel en maak dan die verbinding na 'n gewone telefoon of 'n ander sellulêre telefoon.

Die ontwikkeling van 'n standaard vir GSM sellulêre telefone in al die westerse lande, is in 1982 deur CEPT begin [28, p. 46]. Aangesien sellulêre telefoongebruikers baie rondbeweeg en selfs tussen lande beweeg, is dit belangrik om 'n mate van standardisasie in die sellulêre telefoonbedryf te verkry. Gedurende 1992 het die internasionale kommunikasie beherende outoriteite (CCITT) besluit op 'n standaard wat gebruik sal word. Hierdie standaard staan bekend as die "Group Spécial Mobile" (GSM) [17, p. 62], of later ook as "Global System for Mobile communications" [28, p. 46].

Volgens hierdie standaard word die volgende frekwensies vir die sellulêre stelsel gebruik:

- Basis ontvangs: 890-915 MHz verdeel in 124 kanale
- Basis sending: 935-960 MHz verdeel in 124 kanale
- Mobiele sending: 890-915 MHz verdeel in 124 kanale
- Mobiele ontvangs: 935-960 MHz verdeel in 124 kanale

Om steurings op aangrensende kanale te verhoed, word kanale 1 en 124 gewoonlik nie gebruik nie. In elke band is die kanaalspasiëring vasgestel op 200 kHz en multi tydgleuf toegang (TDMA) met agt tydgleuwe word gebruik.

Frekwensie huppeling word in die mobiele stelsels gebruik. Dit is egter nie die sprekspektrum vorm waar 'n radiofrekwensiesein willekeurig versprei word oor

'n groter bandwydte as wat deur die boodskap benodig word nie. In ware spreispektrum is die bandwydte nie afhanklik van die inhoud van die boodskap nie, maar word bepaal deur die moduleersein. In GSM bly die kanaal nie dieselfde vir die duur van die boodskap nie. Die kanaal verander wanneer daar 'n spesifieke sein ontvang word. Die gebruiker is egter heeltemal onbewus van hierdie kanaalsverandering. Die frekwensie huppeling kom voor tussen die tydgleuwe terwyl die mobiele stel nie ontvang nie. Die hoofrede vir die gebruik van frekwensie huppeling is om te verseker dat die hoogste moontlike vlak van spektrum effektiwiteit verkry word. Die feit dat die boodskap oor 'n aantal kanale gestuur word, help om die uitwerking van 'n swak kanaal te verminder.

2.2.3 Mobiele senders

Mobiele senders word volgens die piekuitset van die telefoon in vyf kategorië verdeel, naamlik: [17, p. 62]

- | | | |
|----------|-----------------------|------|
| • Klas 1 | Voertuig of draagbaar | 20W |
| • Klas 2 | Voertuig of draagbaar | 8W |
| • Klas 3 | Handstel | 5W |
| • Klas 4 | Handstel | 2W |
| • Klas 5 | Handstel | 0.8W |

Gaussian Minimumskuiwleuteling (GMSK) modulاسie, met 'n indeks waarde van 0.3 en 'n dataoordragspoed van 270 kb/s word gebruik. Fase- en frekwensiesinkronisasie moet toelaat vir 'n Doppler verskuiwing op voertuie wat tot 'n snelheid van 250 km/h beweeg. Daar moet ook toegelaat word vir voortsettingsvertraging en heen-en-weer seine tussen die sender en ontvanger in selle tot en met 35 km radius.

Suid-Afrika gebruik 'n digitale stelsel en daarom word die prosessering van spraaktransmissies in digitale terme beskou. Die totale datatempo vir elke radiokanaal is 270 kb/s. Data word in tydsarsies van 577 μ s met 'n data-inhoud van 116 geënkripteerde bisse (Sien Paragraaf 2.2.5), gestuur. Daar is agt tydgleuwe per TDMA raam. Dit wil sê, 'n raam is 'n voorgeskrewe sekwensie van agt sarsies, waar een tydgleuf, aan elke oproeper toegeken word. Agt oproepers word dus gemultiplekseer ten opsigte van tyd.

Tydgleuf	0	1	2	3	4	5	6	7	0	1	2	3	4	5
Funksie		RX			TX		Mon				RX			TX

FIGUUR 2.7 Agt tydgleuwe per TDMA raam

Die seinsterkte van aangrensende selle word gedurig gemonitor om moontlike oorhandiging van een sel na 'n ander te beheer.

2.2.4 Modulasietegniese

Die modulasie wat vir die sellulêre stelsel gebruik word, is Gaussian Minimumskuiwleuteling. Die term 'Gaussian' verwys na die vorm van die moduleer golfvorm. Na die enkripsie eenheid word die bisse na 'n modulator, vir die digitale modulasie van 'n radio frekwensiedraer, oorgedra. Om die modulator in staat te stel om die begin- en stopposisies aan te dui, het dit die vermoë om fopbisse by die datastroom te voeg. Hierdie fopbisse bestaan uit 'n aantal 1's voor en na die datasarsie. Die modulator reageer asof die fopbisse 'n gewone data-inset is. Die bisse word omgeskakel in bipolarêre vorm, met ander

woorde 1 en -1, deur gebruik te maak van die volgende uitdrukkings [28, p. 48] en word gefilter om sodoende die modulasiesein te genereer.

$$q_n = (p_n + p_{n-1}) \bmod 2 \quad (2.1)$$

$$s_n = 1 - 2q_n \quad (2.2)$$

Die laaste uitdrukking verteenwoordig 'n differensiële prosedure, aangesien s_n die verskil word van die bis by posisie n en die huidige posisie. S_n is "Dirac" pulse wat 'n inset is vir 'n filter met spesifieke karakteristieke. Hierdie filter is ontwerp om pulse met 'n Gaussian vorm, te gebruik vir fase-modulasie van 'n radiofrekwensie. Die gestuurde sein in die tyddomein is dus in die volgende vorm [28, p. 48]:

$$x(t) = \sqrt{(2E/T)} \cos[\omega t + \theta(t) + \theta_0] \quad (2.3)$$

waar:

E = energie per modulasiëbis

ω = hoekfrekwensie van die draer

T = tydsduur van een bis

BT (indekswaarde) = 0.3 waar $B = 3$ dB bandwydte

θ = willekeurige faseskuif, wat veronderstel kan word om dieselfde te bly vir die tydsduur van die sarsie.

$\theta(t)$ = die modulasië en produseer 'n faseskuif sowel as sybande in die ongemoduleerde draer.

Die maksimum faseskuif per modulasiëbis is 90° en die modulasië indeks is minder as 0.5.

2.2.5 Kodering vir spraak in sellulêre kommunikasie

GSM is 'n digitale stelsel en daarom moet die spraak omgeskakel word in 'n digitale vorm. Op die oomblik is daar een "codec" wat gebruik word, naamlik RPE-LTP, wat gebruik maak van Lineêre Berekenbare Kodering [28, p. 46]. Hierdie tipe van kodering gee 'n baie goeie weergawe van menslike spraak. 'n Monster word elke 20 millisekondes geneem en 'n blok data van 260 bisse word as volg, met behulp van hierdie monsters, gevorm [28, p. 47]:

Klas 1 (182)	Klas 2 (78)
$d_0 d_1 d_2 \dots d_{181}$	$d_{182} d_{183} d_{184} \dots d_{259}$

FIGUUR 2.8 Blok data van monsters geneem

waar d_0 die mees beduidende bis en d_{259} die mins beduidende bis is. Klas 1 is belangrik aangesien dit 'n reeks van koderingsprosesse deurgaans, terwyl klas 2 onbeskermd bly. Albei klasse vorm gesamentlik 'n nuwe blok data wat die inset na die enkripsie-eenheid sal wees. Die stappe kan as volg opgesom word [28, p. 47]:

Inset →	Klas 1 →	Sikliese →	Permutasie →	Konvolasie →	456
		kode	& stertbisse	kode	
260	182	185	189	378	↑
	Klas 2	_____			
	78				

FIGUUR 2.9 Stappe wat gevolg word om 'n nuwe blok data te vorm

Die sikliese enkodeerder gebruik 'n generator polinoom ($x^3 + x + 1$) om drie kontrolebisse vanaf die polinoom: ($d_0x^{52} + d_1x^{51} + \dots + d_{49}x^3 + c_0x^2 + c_1x + c_2$), te verkry. Die kontrolebisse word so gekies dat die oorblywende

polinoom as volg lyk: $(x^2 + x + 1)$. Indien 'n fout in slegs een bis voor kom, is daar 'n totaal van 53 moontlikhede, maar die kontrolebisse word beperk tot 'n aantal van 8 moontlike kombinasies. Daar moet dus nog prosesse wees om 'n genoegsame vlak van beskerming teen foute te verkry. Die nuwe blok data sal dan as volg hersaamgestel word [28, p. 47]:

0 1 ... 90	91 92 93	94 95 ... 184	185 186 187 188
$d_0 d_2 \dots d_{180}$	$c_0 c_1 c_2$	$d_{181} d_{179} \dots d_1$	
$u_0 u_1 \dots u_{90}$	$u_{91} u_{92} u_{93}$	$u_{94} u_{95} \dots u_{184}$	0 0 0 0

FIGUUR 2.10 Hersamestelling van nuwe blok data

Die ewe getal databisse vanaf 0 tot 180 word in die eerste 91 posisies geplaas. Die onewe getal bisse word in die laaste 91 posisies, in 'n omgekeerde volgorde, geplaas. Daarna volg die vier sterbisse, wat almal 0 is. Die konvolusie enkodeerder genereer 'n nuwe blok volgens die volgende uitdrukkings [28, p. 47]:

$$V_{2n} = (U_n + U_{n-3} + U_{n-4}) \text{ mod } 2 \quad (2.4)$$

$$V_{2n+1} = (U_n + U_{n-1} + U_{n-3} + U_{n-4}) \text{ mod } 2 \quad (2.5)$$

vir $n = 0$ tot 188 en die blok grootte word dus verdubbel. Klas twee bisse word in posisies 378 tot 455 geplaas. Die blok data is nou ten volle saamgestel. Die blokke data wat gestuur moet word, word ingevleg deur die bisse te versprei oor 'n aantal van 8 blokke. Transmissie van 'n normale sarsie is in die vorm van:

Stert	Data	Merker1	Opleidingsorde	Merker 2	Data	Stert	Skerm
3	57	1	26	1	57	3	8.25
	Ge-enkripteerde			Ge-enkripteerde			

FIGUUR 2.11 Normale sarsie

Die effek van die invlegging is dat 57 bisse van die oorspronklike blok, en die oorblywende 57 van die volgende blok afkomstig is. 'n Opleidingsorde word ingesluit tussen die bruikbare databisse. Hierdie 26 bisse word gebruik in die modulasie- en nie in die enkripsie proses nie. Die twee merkers word ingevoeg vir beheerdoeleindes. Vir spraak is die merkers beide zero, maar indien merker 1 of 2 'n '1' is, bevat die ewe- of onewe bisse beheerinligting. 'n Vol blok data bevat dus 456 bisse en 8 merkers, wat 'n totale grootte van 464 bisse gee.

GSM bied dus 'n groot mate van sekuriteit, al sou enkripsie nie gebruik word nie, aangesien dit van baie komplekse seinrangskikkings, frekwensie huppeling en multipleksering van boodskappe gebruik maak.

2.2.6 Kodering vir dataoordrag in sellulêre kommunikasie

GSM is nie beperk tot slegs spraak nie, maar het ook fasiliteite vir die stuur van data teen die volgende tempo's:

- (a) Vol tempo teen 9600 kbis/sekonde
- (b) Vol tempo teen 4800 kbis/sekonde
- (c) Vol tempo teen 2400 kbis/sekonde
- (d) Half tempo teen 4800 kbis/sekonde
- (e) Half tempo teen 2400 kbis/sekonde

In geval (a) is die inset 60 bisse elke vyf millisekondes. Vier blokke word saam gegroepeer om een 240 bis blok te vorm, wat as volg geprosesseer word [28, p. 47]:

Inset→	4 blokke→	Stert→	Konvolasie kode→	Vermindering na
60	240	244	488	456
				456

FIGUUR 2.12 Vier blokke data van 60 bisse elk

Die konvolasie kode maak gebruik van dieselfde twee uitdrukkings as wat vir spraak gebruik word, naamlik (Sien Paragraaf 2.2.5):

$$V_{2n} = (U_n + U_{n-3} + U_{n-4}) \bmod 2 \quad (2.6)$$

$$V_{2n+1} = (U_n + U_{n-1} + U_{n-3} + U_{n-4}) \bmod 2 \quad (2.7)$$

Die bisse in posisies 11 tot 42 word uitgehaal, om sodoende 'n verkleinde blok van 456 bisse te verkry. Hierdie blok word dan ingevleg om sodoende oor 19 blokke versprei te word.

In geval (b), met ander woorde voltempo teen 4.8 kbisse/sekonde, is die inset 60 bisse elke tien millisekonde. Die prosedure kan in Figuur 2.13 gesien word [28, p. 47]:

Inset→	Stert →	2 blokke→	Konvolasie kode →	Invlegging
60	76	152	456	

FIGUUR 2.13 60 bisse elke 10 millisekonde

Die stertbisse word as volg ingesluit in die 76-bis blok:

Data	Stert	Data	Stert	Data	Stert	Data	Stert
d ₀ -d ₁₄	bisse	d ₁₅ -d ₂₉	bisse	d ₃₀ -d ₄₄	bisse	d ₄₅ -d ₅₉	bisse
15	4	15	4	15	4	15	4
u ₀							u ₇₅

FIGUUR 2.14 Invoeging van stertbisse

'n Verdere blok, soos bogenoemde, word gegeneer en in posisies 76 tot 151 geplaas. Die konvolasie kode verdriedubbel dus die blok tot 456 bisse, met behulp van onderstaande vergelykings [28, p. 47]:

$$V_{3n} = (U_n + U_{n-1} + U_{n-3} + U_{n-4}) \bmod 2 \quad (2.8)$$

$$V_{3n+1} = (U_n + U_{n-2} + U_{n-4}) \bmod 2 \quad (2.9)$$

$$V_{3n+2} = (U_n + U_{n-1} + U_{n-2} + U_{n-3} + U_{n-4}) \bmod 2 \quad (2.10)$$

en word ingevleg, ge-enkripteer en gestuur in die formaat van 'n normale sarsie data.

2.2.7 Enkripsietegnieke in sellulêre stelsels

Vir data en spraak word die enkripsieproses gedoen na die enkodering en net voor die modulasieproses.

Die enkripsie- en de-kripsieprosedures is egter nie gepubliseer as normale spesifikasies nie. Die algoritme is beperk tot vervaardigers en is slegs beskikbaar aan sekere uitgesoekte personeel van hierdie firmas [28, p. 49].

2.2.8 Voordele van 'n sellulêre stelsel

- Die belangrikste voordeel van die sellulêre netwerk is die grootte van die toerusting wat benodig word. Die toerusting is baie klein en is dus ideaal vir die gebruik in mobiele kommunikasietoepassings.
- Sellulêre telefone gebruik herlaaibare batterye en geen spesiale kragbron of ander toerusting word benodig nie.
- Alhoewel die modems baie duur is, ongeveer R3000 elk, is dit nog steeds goedkoper as 'n satellietkoppeling. Sellulêre dekking sal ook in die toekoms verbeter en weldra sal die beskikbaarheid van datakommunikasietoerusting verbeter, soos wat die vraag daarna verhoog.
- Indien dekking in 'n gebied beskikbaar is, word geen verdere herleistasies benodig nie.
- Koppeling met die publieke skakelnetwerk word gemaak en die aantal bestemmings vir sellulêre telefoonoproepe is dus nie beperk tot slegs ander sellulêre telefone nie.
- Antennes is baie klein. Daar is ook reeds klein draagbare antennes beskikbaar om ontvangs te verbeter.
- Die meeste sellulêre telefone is ontwerp vir dieselfde werking as gewone telefone, en is dit dus baie maklik om die toerusting te gebruik.

2.2.9 Nadele van 'n sellulêre stelsel

- Tans vorder die dekking in die Vrystaat baie stadig en is daar nog groot dele waar sellulêre dienste nie beskikbaar is nie (Sien Bylaag 1).
- Modems vir sellulêre telefone is huidiglik baie duur.
- Seinfaling kan ook 'n probleem wees in veraf gebiede en ook in gebiede waar ontvangs baie sleg is, soos in sommige geboue.

2.3 Satellietkoppelings

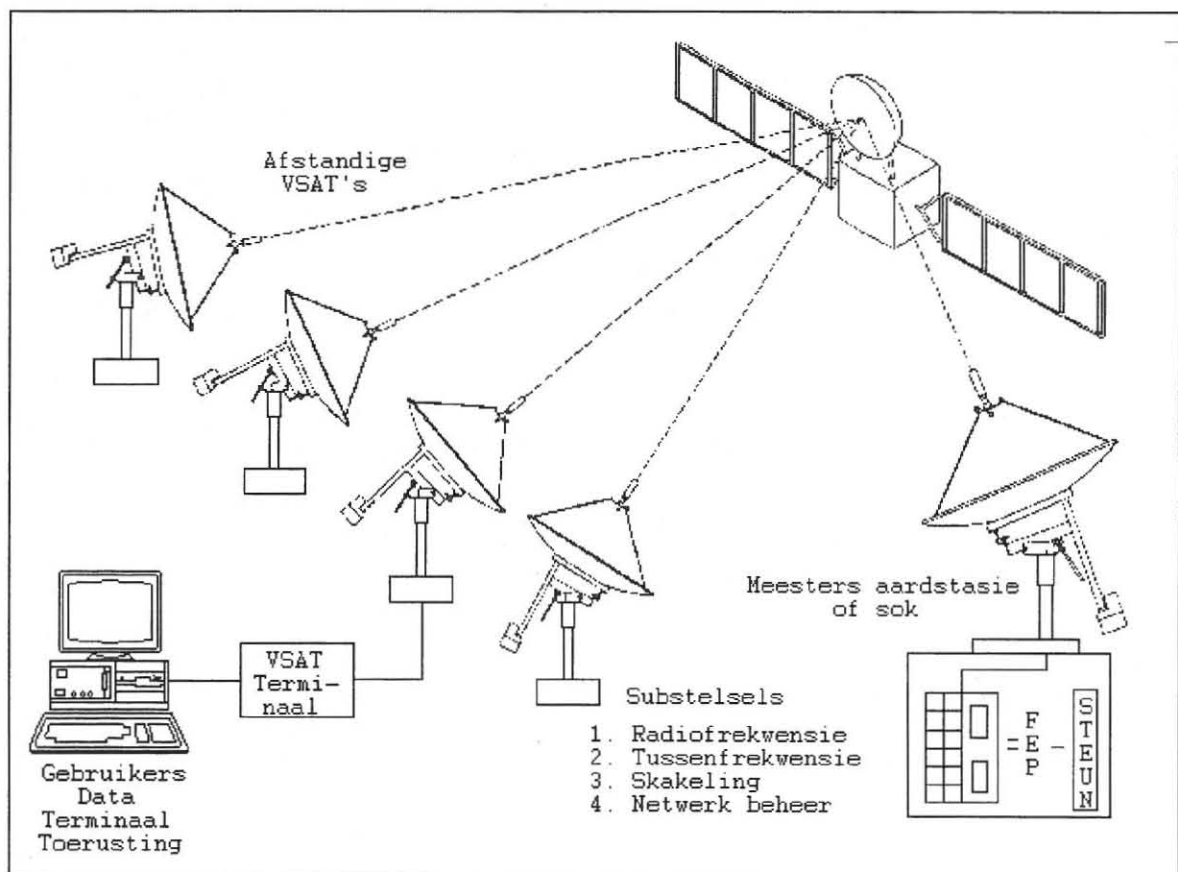
Satelliet gebaseerde netwerke het 'n beskeie begin in die vroeë 1980's gehad [5, p. 64], maar vandag bied dit 'n oplossing vir wye-gebiedkommunikasie vir korporasies regoor die wêreld. "Very Small Aperture Terminal" (VSAT) netwerke word tans beskou as die mees buigsame tipe vir kommunikasiedoeleindes. Dit is alreeds die kommunikasiemiddel wat baie afgeleë gebiede met 'n sentrale rekenaar verbind [5, p. 64].

Frank Booty skryf dan ook in sy artikel dat die sterkonfigurasië van 'n VSAT-netwerk 'n doeltreffende, sowel as koste-effektiewe metode bied, vir die oorsending van data, beeldmateriaal, spraak en fakse deur 'n wydverspreide organisasie, teen dieselfde betroubaarheidsvlakke by enige van die aardstasies, ongeag van posisie of ligging.

Die wêreldwye mark vir VSAT-toerusting en dienste word, in Desember 1994, op 'n bedrag van \$350 miljoen per jaar geraam. Hierdie industrie ondervind 'n geweldige groei vanaf die 1980's en markanalise het getoon dat dit sal stabiliseer teen 20% groei per jaar deur die 1990's. Tans is daar meer as 100000 VSAT's reg oor die wêreld geïnstalleer, of reeds op bestelling. 'n Groot deel van die groei kan toegeskryf word aan 'n hoë vlak van klante tevredenheid ten opsigte van die diens wat gelewer word. Baie gebruikers prys hierdie VSAT-netwerke vir verhoogde netwerk beskikbaarheid, asook verbeterde beheer oor die firma se kommunikasiekostes.

2.3.1 Werking van VSAT-stelsels

'n VSAT-netwerk bestaan uit 'n sentrale meester-aardstasie of sok en geografies verspreide VSAT's op aarde. Sien Figuur 2.15 [5, p. 66]. Die sok bestaan uit 'n satellietantenne (5 tot 9 meter in deursnee), radio frekwensie- en tussenfrekwensie toerusting, 'n netwerkbeheerstelsel en 'n skakelstelsel wat aan die kliënt se hoofraamrekenaar gekoppel word. Die skakelstelsel is die hart van die sok en dien as roetwyser vir die data vanaf enige punt in die netwerk na 'n ander. Die netwerkbeheerstelsel stel die netwerkhanteerder in staat om die netwerk se konfigurasie te verander, data af te laai, die netwerk te monitor en foutsporing te kan doen. Al hierdie funksies en veranderings kan deur die netwerkhanteerder vanaf 'n sentrale sokposisie gedoen word [5, p. 65].



FIGUUR 2.15 'n VSAT stelsel

Die geografies verspreide VSAT's op aarde, bestaan uit klein, maklik installeerbare antennes. Hierdie antennes is ongeveer een tot twee meter in deursnee. Verder word 'n buitelug RF eenheid (ODU), asook 'n binne digitale prosesseringseenheid (IDU) benodig. Die IDU is verantwoordelik vir kommunikasiefunksies wat verband hou met die versending en ontvangs van data. Normaalweg sal daar twee tot vier RS232 poorte op die eenheid beskikbaar wees, wat tot drie protokolle gelyktydig kan steun. Hierdie eienskap stel dus die gebruiker in staat om verskeie dataterminaltoerusting aan die netwerk te koppel.

Die installering van 'n VSAT-afstandstasie is 'n eenvoudige drie-stap proses:

1. Bestudering van die plaaslike reglement, plaaslike lisensie- en permit vereistes.
2. Antenne geïnstalleer om 'n ononderbroke ("line of sight") sein vanaf die satelliet te verkry.
3. Kommunikasie word bewerkstellig tussen die sok en die gebruiker se rekenaar om die funksionering van die stelsel te evalueer.

Om data te ontvang of te versend tussen die sok en 'n afstandstasie word die data na 'n geostasionêre Ku- of C-Band satelliet gestuur. Hierdie geostasionêre satelliete wentel op 'n hoogte van 35800km bokant die ewenaar en voltooi een omwenteling in 24 uur [27]. Aangesien hierdie satelliete teen dieselfde hoeksnelheid as die aarde wentel, bly hulle posisie ten opsigte van die antenne op aarde konstant. Dit is dus nie nodig om die satelliet te volg of die antenne te herposisioneer nie.

2.3.2 Frekwensiebande wat gebruik word vir satellietkommunikasie

Daar bestaan verskeie bande wat toegestaan is vir satellietkommunikasie. Tabel 2.1 [32, p. 400] dui die verskillende bande met die frekwensies wat vir elke band gebruik word aan.

Hoe hoër die frekwensie wat gebruik word, hoe hoër is die koste van die toerusting wat benodig word, aangesien lae frekwensieseine minder verswak word. Om hierdie rede is die afskakel gewoonlik ook die laer frekwensie van die twee, aangesien 'n laer kraguitset van die satellietsender gebruik kan word.

TABEL 2.1 Kommunikasiesatelliet frekwensietoekennings (MHz)

Gebruik	Afskakel frekwensie (MHz)	Opskakel frekwensie (MHz)
	Vaste dienste	
Kommersiëel (C-band)	3700 - 4200	5925 - 6425
Militêr (X-band)	7250 - 7750	7900 - 8400
Kommersiëel (K-band)	11700 - 12200	14000 - 14500
	10950 - 11200	27500 - 31000
	11450 - 11700	
	17700 - 21200	
	Mobiele dienste	
Maritiem	1535 - 1542.5	1635 - 1644
Lugvaart	1543.5 - 1558.8	1645 - 1660
	Uitsaai dienste	
	2500 - 2535	2655 - 2690
	11700 - 12750	

2.3.3 Voordele van VSAT-stelsels

- Die grootste voordeel van 'n VSAT is die wye dekking wat die stelsel bied. Kommunikasie is dus moontlik in bykans enige deel van die Republiek van Suid-Afrika.
- Relatiewe hoë datatempo's is moontlik met behulp van 'n satellietverbinding.
- Indien 'n VSAT stelsel gebruik word, sal kommunikasie 24 uur per dag beskikbaar wees.
- Geen herleistasies word benodig nie.

2.3.4 Nadele van VSAT-stelsels

- Tans is dit slegs Telkom wat regtens hierdie dienste in Suid Afrika mag verskaf. Daar word ongeveer 120 mobiele klinieke slegs in die Vrystaat benodig. Die huur van een VSAT afstandstasie het in 1995 ongeveer R4 000 per maand beloop. Indien op 'n basis van 120 mobiele VSAT's gewerk word, sal die maandelikse bedryfskoste van so 'n stelsel ongeveer R480 000 beloop. Hierdie bedrag, tesame met die kapitaal insetkoste van die rekenaars en sagteware wat in die mobiele eenhede gebruik gaan word, sal meebring dat hierdie stelsel nie 'n koste-effektiewe oplossing vir die probleem, sal bied nie.
- Die toerusting is duur.
- Die toerusting is groot. 'n Sleepwa sal saam met die mobiele kliniek gesleep moet word, om die sender, ontvanger en antenne te vervoer.
- Antennes is groot (5 tot 9 meter in deursnee).

2.4 Radioverbindinge

Die mees algemene vorm van datakommunikasie op 'n radiokanaal, staan bekend as pakketradio. Pakketskakeling is 'n vorm van datakommunikasie wat data oordra deur dit in kleiner pakkies te verdeel.

Pakketradio is deur die Universiteit van Hawaii in 1970 in gebruik geneem, vir die oordrag van data na afgeleë kampusse, wat oor die eilande versprei was. Hierdie stelsel het bekend gestaan as ALOHANET en kan gesien word as die direkte voorloper van amateur pakketradio [19, p. 2-1].

2.4.1 Werking van radiotoerusting en netwerke

Gewoonlik word 'n simpleks- of 'n half-duplekskanaal gebruik saam met pakketradio, maar by vinniger stelsels kan 'n duplekskanaal gebruik word. Wanneer 'n dupleksstelsel gebruik word, moet daar gebruik gemaak word van twee frekwensies vir die daarstelling van een kanaal. Dit is dus 'n baie ingewikkelder proses as wanneer simpleksstelsels gebruik word, aangesien die ontvanger en sender terselfdertyd moet werk [2, p. 19-30].

'n Modem word gebruik om 'n digitale basisbandsein na 'n analogsein om te skakel, vir die oordrag van die data oor 'n analogkanaal. Analog kommunikasiemediums het gewoonlik 'n basisband bandwydte van 3-4kHz. 'n Verwantskap bestaan tussen hierdie bandwydte en die datatempo's wat verkry kan word. 'n 3-4kHz analogkanaal kan datatempo's van 1200 bisse/sekonde, maklik haal, terwyl datatempo's van 9600 bisse/sekonde met die gebruik van komplekse modulasiemetodes gehaal kan word. Hierdie hoë snelhede kan slegs

met 'n hoë sein- tot ruisverhouding verkry word. Indien die kanaal nie van goeie gehalte is nie, moet daar teruggeval word na stadiger datatempo's [2, p. 19-30].

2.4.2 Modulasietegnieke vir pakketradio

Binêre frekwensieskuifsleuteling (FSK) is die mees algemene tipe modulasie wat gebruik word in pakketradio. Hierdie tipe modulasie maak gebruik van 'n binêre '1' - 'n toon van byvoorbeeld 1200Hz en van 'n binêre '0' 'n toon van 2200Hz. Minimumskuifsleuteling (MSK) is 'n vorm van FSK, waar die frekwensieskuif in hertz die helfte van die seinspoed in bauds is. Met ander woorde, 'n 1200-baudsein sal met 600Hz geskuif word [2, p. 19-30].

Faseskuifsleuteling (PSK) is 'n ander vorm van modulasie wat gebruik word vir datakommunikasie. Vir amateurradio is die gebruik van hierdie tipe modulasie beperk tot satelliet- en laesnelheid aardstransmissies, maar belangstelling groei vir die gebruik van hierdie tipe modulasie in hoë snelheid pakketradio. Terwyl binêre faseskuifsleuteling baie betroubaar is, is dit nie baie spektrum-effektief nie. M-ary PSK, soos byvoorbeeld 4PSK en 8PSK laat transmissies van meer bisse per sekonde, teen 'n laer baudspoed, toe [2, p. 19-30].

Hoewel slegs 'n paar van die belangrikste modulasietegnieke bespreek is, is daar nog baie ander wat gebruik kan word [40].

2.4.3 Voordele van radiokommunikasie

- Vir hierdie stelsels is dit nie altyd nodig om 'n lyn van sig te hê, soos by satellietstelsels nie.

- Hierdie stelsel is relatief goedkoop teenoor 'n sellulêre stelsel en ook 'n satellietstelsel, aangesien geen oproepkoste betaal word nie, en die toerusting wat benodig word, nie duur is nie.
- Die toerusting benodig vir hierdie stelsel is groter as die vir sellulêr en kleiner as die vir VSAT's.
- Antennes is klein en kan maklik op 'n voertuig monteer word.
- Kragbron klein (Voertuig).

2.4.4 Nadele van radiokommunikasie

- Die grootste nadeel van 'n radiostelsel is, dat herleistasies benodig word vir die dekking van alle gebiede.
- Aangesien dekking beperk sal wees tot gebiede waar herleistasies bestaan is dekking beperk.

2.5 Keuse van die kommunikasiemedium

Vir hierdie projek is dit belangrik om 'n kommunikasiemedium te kry wat van klein toerusting gebruik maak, sodat dit geskik sal wees vir gebruik in mobiele eenhede. Dit is ook belangrik dat die toerusting vanaf die voertuig se battery moet kan werk, sodat die stelsel ten volle mobiel kan wees. Die onderstaande tabel, Tabel 2.1, dui die kriteria waarvolgens die keuse gemaak is, met verwysing na die verskillende stelsels.

Alhoewel satelliet dekking baie goed is, is satelliet toerusting baie groot en duur en dus nie 'n koste-effektiewe oplossing vir die probleem nie.

- Hierdie stelsel is relatief goedkoop teenoor 'n sellulêre stelsel en ook 'n satellietstelsel, aangesien geen oproepkoste betaal word nie, en die toerusting wat benodig word, nie duur is nie.
- Die toerusting benodig vir hierdie stelsel is groter as die vir sellulêr en kleiner as die vir VSAT's.
- Antennes is klein en kan maklik op 'n voertuig monteer word.
- Kragbron klein (Voertuig).

2.4.4 Nadele van radiokommunikasie

- Die grootste nadeel van 'n radiostelsel is, dat herleistasies benodig word vir die dekking van alle gebiede.
- Aangesien dekking beperk sal wees tot gebiede waar herleistasies bestaan is dekking beperk.

2.5 Keuse van die kommunikasiemedium

Vir hierdie projek is dit belangrik om 'n kommunikasiemedium te kry wat van klein toerusting gebruik maak, sodat dit geskik sal wees vir gebruik in mobiele eenhede. Dit is ook belangrik dat die toerusting vanaf die voertuig se battery moet kan werk, sodat die stelsel ten volle mobiel kan wees. Die onderstaande tabel, Tabel 2.1, dui die kriteria waarvolgens die keuse gemaak is, met verwysing na die verskillende stelsels.

Alhoewel satelliet dekking baie goed is, is satelliet toerusting baie groot en duur en dus nie 'n koste-effektiewe oplossing vir die probleem nie.

Radio as medium het geen oproepkoste nie, maar herleistasies word benodig, wat die aanvangskapitaal baie verhoog. Hoewel sellulêre dekking in die Vrystaat op die oomblik nie so goed is nie, kan verwag word dat die dekking baie sal verbeter, aangesien die sellulêre netwerk 'n nuwe ontwikkeling in Suid-Afrika is.

Vir bogenoemde redes is daar dus besluit om sellulêr as medium te gebruik.

TABEL 2.2 Vergelyking van verskillende mediums volgens kriteria

Kriteria	Kommunikasiemedium		
	Sellulêr	Satelliet	Radio
Fisiese grootte van toerusting	Klein (draagbaar)	Sleepwa benodig vir voertuig	Klein
Kragbron vereistes	Herlaaibare batterye, sowel as die van die voertuig	Voertuig battery	Voertuig battery
Koste	Klein uitlegkoste, maar hoë oproepkoste	Toerusting duur	Toerusting ongeveer dieselfde as sellulêre telefone, maar geen oproepkoste
Tyd beskikbaar vir kommunikasie	24 uur per dag	24 uur per dag (Indien VSAT)	24 uur per dag
Word herleistasies benodig	Nee Torings deur diensverskaffers	Nee	Ja
Grootte van dekking	Sien Bylaag 1	Hele Suid-Afrika	Afhangend van aantal herleistasies
Antennes	Klein (Voertuig gemonteer sowel as draagbaar)	5 tot 9 meter in deursnee	Klein (Voertuig gemonteer)

2.6 Opsomming

Hierdie hoofstuk het gehandel oor die gebruik van 'n langafstand fisiese kommunikasiemedium wat gebruik kan word in mobiele kommunikasie vir

afgeleë gebiede. Kommersiële kommunikasienetwerke soos sellulêre telefone, satellietstelsels en verskillende frekwensie radiotransmissiestelsels is teen mekaar opgeweeg en op grond van 'n prioriteitskriteria gedefineer in Paragraaf 2.1. Hoewel sellulêre dekking nie op die stadium voldoende is vir dekking na alle afgeleë gebiede nie, is die beplanning binne die afsienbare toekoms verseker. Verder is die toerusting klein, kompak en relatief goedkoop tot die ander kommunikasiemedia wat teen mekaar opgeweeg is.

Die stelsel word dus ontwikkel vir sellulêre datakommunikasie wat feitlik identies op radio- en satellietkanale gebruik sal kan word. Die volgende hoofstuk handel dus oor die gebruik van hierdie fisiese medium in 'n logiese netwerkhierargiestruktuur ten einde 'n wye areanetwerk (WAN) te vestig.

HOOFSTUK 3

REKENAARNETWERKE

Die vorige hoofstuk het sellulêre datakommunikasie as fisiese medium vir die vestiging van 'n WAN gekies. In hierdie hoofstuk word 'n uiteensetting gegee van die verskillende netwerke, asook die verskillende netwerkprotokolle wat oorweeg is vir die projek. Paragraaf 3.1 gee 'n oorsig van verskillende netwerke, terwyl Paragraaf 3.2 handel oor die verskillende protokolle en ook die redes waarom daar besluit is om TCP/IP as die protokol vir hierdie projek te kies.

3.1 Rekenaarnetwerke

In rekenaarterme bestaan 'n netwerk uit 'n groep rekenaars en verwante toestelle wat verbind is deur kommunikasiefasiliteite. 'n Netwerk kan bestaan uit permanente verbindings soos kables, of tydelike verbindings wat gemaak word deur middel van telefoonkoppelings of ander kommunikasiemediums [27]. 'n Netwerk kan bestaan uit slegs 'n paar rekenaars, drukkers en ander aparatuur (LAN) of dit kan bestaan uit 'n aantal groot of klein rekenaars wat versprei is oor 'n groot geografiese area (WAN).

Netwerke het die vermoë om inligting, wat andersins baie lank sou neem om te versamel, aan 'n persoon beskikbaar te stel in die gerief van sy eie huis of kantoor. 'n Voorbeeld hiervan is die Internet, 'n versameling van netwerke, wat 'n wêreldwye netwerk daarstel. Informasie of elektroniese pos kan dus binne 'n paar minute van een deel van die wêreld na 'n ander, duisende kilometers daarvandaan, versend word.

Daar is hoofsaaklik drie redes vir die gebruik van 'n netwerk [16, p. 2], nl:

1. *Afstandige aantekening*: Hierdie funksie stel 'n netwerkgebruiker in staat om vanaf 'n ander plek aan te teken, en die bronne van die netwerk te gebruik.
2. *Lêer oordrag*: Hierdie opsie stel gebruikers in staat om lêers uit te ruil en duplikasie van lêers te voorkom. Dit is dus 'n gerieflike en betroubare stelsel.
3. *Elektroniese pos*: Die funksie maak dit vir gebruikers moontlik om elektroniese pos, met ander woorde boodskappe en lêers, aan mekaar te stuur. Dit is 'n maklike vinnige manier van inligtingoordrag terwyl dit geld en tyd bespaar. In hierdie geval word 'n boodskap by die sender se bediener gelaat om verder gepos te word. Die betrokke bediener stuur dan die boodskap na die ontvanger se bediener wat dit op sy beurt beskikbaar stel vir die ontvanger sodra hy aanteken op sy netwerk.

In 'n tipiese rekenaarnetwerkkonfigurasie word twee of meer rekenaars aan mekaar verbind deur middel van 'n fisiese koppeling, met 'n logiese netwerk en sy protokol daarop gesuperponeer. In 'n lokale areanetwerk (LAN) is die koppeling meestal deur middel van koaksiale kabel [16, p. 11], optiese vesel of koper kabel. In 'n wye areanetwerk (WAN) kan die fisiese koppeling deur middel van mikrogolf, satelliet, radio of telefoon wees. In hierdie gevalle is die koppeling meestal seriaal van aard [19, p. 3-3].

Die netwerkbediener vorm die kern van die rekenaarnetwerk wat dan 'n aantal onafhanklike rekenaars (meestal persoonlike rekenaars) bedien. Die netwerkbediener is in die meeste gevalle ook 'n persoonlike rekenaar wat voorsien is van 'n netwerkkaart en kommersiële netwerkbedienersagteware (soos byvoorbeeld Novell) met 'n unieke adres. 'n Gas-rekenaar, wat voorsien is

van 'n netwerkkaart met 'n sagtewaredrywer vir die spesifieke kommersiële bedienersagteware (ook argumentsonthalwe Novell), word dan aan die betrokke bediener, in 'n LAN of WAN konfigurasie gekoppel, en ook voorsien van 'n unieke adres om op die betrokke netwerk te kommunikeer.

3.2 Netwerkprotokolle

In die hoër vlakke van die netwerkhierargie, wat later in meer detail behandel sal word, word die dataoordrag gereguleer deur middel van die netwerkprotokol [10, p. 13]. Hierdie protokol sien toe dat die data reg oorgedra word, dat foutkorreksie plaasvind en dat die datapakette korrek by die ontvanger saamgestel word.

3.2.1 Die funksie van 'n netwerkprotokol

'n Netwerkprotokol bepaal alle bewerkings wat binne 'n netwerk uitgevoer moet word. Protokolle bepaal verder ook die wisselwerking tussen eenhede, buite sowel as binne die netwerk. 'n Voorbeeld hiervan is, waar party netwerkprotokolle bepaal hoe data vanaf punt A na punt B gestuur sal word, terwyl ander netwerkprotokolle bepaal hoe toerusting of 'n rekenaar oor 'n spesifieke medium, byvoorbeeld 'n telefoonlyn, sal kommunikeer [16, pp. 1, 2].

3.2.2 TCP/IP as netwerkprotokol

TCP/IP ("Transmission Control Protocol / Internet Protocol") is 'n reeks protokolle wat ontwikkel is om rekenaars, wat aan 'n netwerk gekoppel is, toe

van 'n netwerkkaart met 'n sagtewaredrywer vir die spesifieke kommersiële bedienersagteware (ook argumentsonthalwe Novell), word dan aan die betrokke bediener, in 'n LAN of WAN konfigurasie gekoppel, en ook voorsien van 'n unieke adres om op die betrokke netwerk te kommunikeer.

3.2 Netwerkprotokolle

In die hoër vlakke van die netwerkhiërargie, wat later in meer detail behandel sal word, word die dataoordrag gereguleer deur middel van die netwerkprotokol [10, p. 13]. Hierdie protokol sien toe dat die data reg oorgedra word, dat foutkorreksie plaasvind en dat die datapakette korrek by die ontvanger saamgestel word.

3.2.1 Die funksie van 'n netwerkprotokol

'n Netwerkprotokol bepaal alle bewerkings wat binne 'n netwerk uitgevoer moet word. Protokolle bepaal verder ook die wisselwerking tussen eenhede, buite sowel as binne die netwerk. 'n Voorbeeld hiervan is, waar party netwerkprotokolle bepaal hoe data vanaf punt A na punt B gestuur sal word, terwyl ander netwerkprotokolle bepaal hoe toerusting of 'n rekenaar oor 'n spesifieke medium, byvoorbeeld 'n telefoonlyn, sal kommunikeer [16, pp. 1, 2].

3.2.2 TCP/IP as netwerkprotokol

TCP/IP ("Transmission Control Protocol / Internet Protocol") is 'n reeks protokolle wat ontwikkel is om rekenaars, wat aan 'n netwerk gekoppel is, toe

te laat om bronne, met ander woorde sagteware en data, te kan deel. Dit is daargestel deur 'n groep navorsers, wat nou betrokke was by die ontwikkeling van ARPAnet, wat moontlik die mees bekende TCP/IP netwerk is. Sedert Junie 1987 het ten minste 130 verskillende verskaffers van sagteware, produkte bekend gestel wat die TCP/IP protokol gebruik [16, p. 1].

Alhoewel daar netwerke is wat as 'n TCP/IP netwerk bekend staan, is dit nie noodsaaklik dat die netwerk van TCP sowel as IP gebruik maak nie. Daar bestaan dus netwerke wat net van die IP protokol gebruik maak.

Internet is 'n versameling van netwerke wat aan mekaar gekoppel is en wat gebruik maak van IP. TCP/IP is 'n familie van protokolle waarvan 'n paar soos byvoorbeeld IP, TCP, UDP laevlak protokolle is, wat deur verskeie toepassingsagteware gebruik word. Daar bestaan wel ook ander protokolle wat vir spesifieke take ontwikkel is, wat take soos die oordrag van lêers tussen rekenaars, die stuur van elektroniese pos en die vasstelling van alle netwerkgebruikers, insluit. Aanvanklik is TCP/IP meestal tussen minirekenaars of hoofraamrekenaars, wat hardeskywe gehad het en onafhanklik kon werk, gebruik.

'n Belangrike eienskap van die TCP/IP protokol is die voorsiening vir seriale datakommunikasie deur die seriale RS232-koppelvlak van 'n persoonlike rekenaar. 'n Rekenaar netwerkkaart word dus nie gebruik nie, maar wel 'n "Serial Link Internet Protocol" (SLIP) drywer deur middel van sagteware onderbrekers, wat die dataoordrag deur die seriepoort beheer [19, p. 11-26].

3.2.3 Die plasing van TCP/IP in die netwerkhierargie

Indien betroubare dataoordrag tussen twee rekenaars verlang word, is daar 'n verskeidenheid van prosedures wat gevolg moet word, nl. [18, p. 180]:

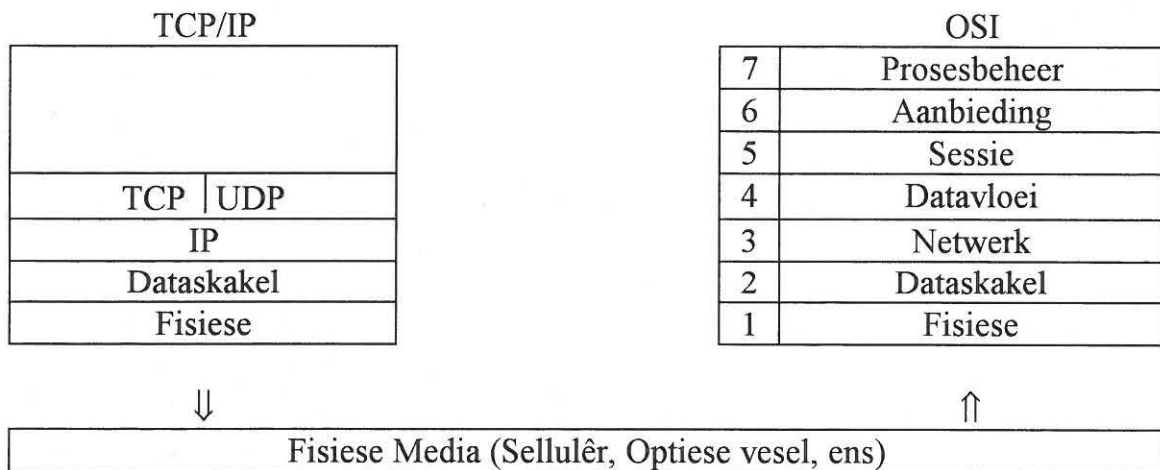
1. Formateer data.
2. Opmaak van data in pakkette.
3. Die vasstelling van dataroete.
4. Regulering van die tempo van dataversending, na gelang van die beskikbare bandwydte en die kapasiteit van die ontvanger om data te absorbeer.
5. Versending van data op die fisiese medium.
6. Samestelling van inkomende data sodat dit in volgorde is en geen dele daarvan kortkom nie.
7. Kontrolering van inkomende data vir duplisering.
8. Erkennung aan die sender van die hoeveelheid data wat foutloos ontvang is.
9. Lewering van data aan die regte toepassing.
10. Hantering van foute en probleme.

Die eindresultaat hiervan is, dat kommunikasiesagteware ingewikkeld is. Om hierdie probleem te help oplos, word kommunikasiesagteware 'n struktuur gegee, wat maklik verstaanbaar sowel as veranderbaar is. Die "International Standards Organization" (ISO) het dan die "Open Systems Interconnection" (OSI) Verwysingsmodel ontwikkel [10, p.6]. Hierdie model word algemeen gebruik en kan bykans op enige kommunikasie-sagteware van toepassing gemaak word. Figuur 3.1 toon die verwysings model:

Prosesbeheer
Aanbieding
Sessie
Datavloei
Netwerk
Dataskakel
Fisiese

FIGUUR 3.1 OSI Netwerkhierargie verwysingsmodel

Indien ons die OSI Verwysingsmodel gebruik, kan ons sien waar TCP/IP in hierdie model pas [10, p. 9]. Sien onderstaande figuur:



FIGUUR 3.2 TCP/IP in die OSI verwysingsmodel

3.2.4 Beskrywing van TCP/IP protokolle

TCP/IP is 'n reeks protokolle wat uit lae bestaan [16, p. 5]. Ter verduideliking kan lêeroordrag as voorbeeld gebruik word. Eerstens is daar 'n protokol vir lêeroordrag, naamlik FTP ("File Transfer Protocol"), wat 'n stel reëls neerlê, waarvolgens een rekenaar aan 'n ander 'opdragte' gee. Dit sluit inligting soos die sender van die lêer, die ontvanger van die data en die data self in. Die protokol neem egter aan dat daar 'n manier is om betroubaar tussen die twee rekenaars te

kommunikeer. Daar moet dus 'n fisiese verbinding tussen die twee rekenaars bestaan om betroubare oordrag van die data te verseker.

Lêeroordrag, soos baie ander toepassingsprotokolle, definieer dus slegs 'n stel opdragte en boodskappe wat gestuur moet word. Dit word dus ontwerp om saam met ander protokolle soos byvoorbeeld TCP en IP gebruik te word.

TCP is verantwoordelik om te verseker dat die opdragte oorgedra word na die ander rekenaar en hou terselfertyd rekord van wat versend en ontvang is deur die ander rekenaar. TCP sal die data hersend, indien dit nie reg ontvang is nie. Indien 'n lêer te groot is vir een datagram, sal die data deur TCP opgedeel word in verskillende datagramme, en sal ook seker gemaak word dat die data reg oorgedra word na die ander rekenaar. Aangesien hierdie funksies deur baie ander toepassings gebruik sal word, word dit dus in 'n aparte protokol gedefinieer, in plaas van om dit in een protokol te omvat. Daar kan dus aan TCP gedink word as 'n biblioteek van roetines wat deur verskillende toepassings, indien nodig, gebruik kan word om 'n betroubare netwerkverbinding met 'n ander rekenaar te bewerkstellig.

TCP maak op sy beurt weer gebruik van IP, maar alhoewel die dienste wat TCP lewer, deur baie toepassings gebruik word, is daar ook toepassings wat nie van TCP gebruik maak nie. Daar is egter dienste wat deur alle toepassings benodig word. Hierdie dienste word dus in die IP protokol gedefinieer. Soos by TCP, kan ons dus ook die voorbeeld van die biblioteek gebruik. IP vorm dus 'n biblioteek van roetines wat deur TCP, sowel as toepassings wat nie van TCP gebruik maak nie, gebruik word. Hierdie manier van protokolle wat in lae saam gebruik word, word dan ook "layering" genoem. TCP en IP is dus verskillende lae wat elkeen van die dienste van die laag onder hom gebruik maak.

TCP/IP toepassings maak gewoonlik gebruik van vier lae:

1. 'n Toepassings protokol soos FTP.
2. 'n Protokol soos byvoorbeeld TCP, wat dienste verskaf aan baie toepassings.
3. IP, wat die basiese dienste, soos byvoorbeeld die besorging van data aan hul regte bestemmings, verskaf.
4. Die protokolle wat benodig word deur die fisiese vlak, soos byvoorbeeld Ethernet.

TCP/IP veronderstel ook dat daar 'n groot aantal onafhanklike netwerke deur middel van deurgange aan mekaar gekoppel is. Die gebruiker moet dus in staat wees om enige rekenaar te kan bereik en enige bronne van ander netwerke te kan deel. Datagramme sal soms deur 'n dosyn verskillende netwerke beweeg voordat die eindbestemming bereik word. Die gebruiker moet ook nie gemoeid wees met die roetebepaling wat benodig word vir die versending van die datagramme nie. Sover dit die gebruiker aangaan, moet dit slegs nodig wees om die Internet adres van die ander rekenaar te weet. Hierdie adres kan as volg lyk: 198.54.58.1, wat 'n 32-bis getal is. Dit word gewoonlik as vier desimale getalle, wat elkeen agt bisse verteenwoordig, geskryf. Die term oktet in plaas van woord, word gewoonlik in terme van TCP/IP gebruik, aangesien TCP/IP ook deur stelsels met ander woordgroottes as agt bisse, gebruik word.

Die struktuur van die adres gee 'n bietjie inligting van die stelsel, byvoorbeeld 198.54.58 is 'n netwerknommer wat deur die sentrale outoriteit (InterNIC) aan Technikon Vrystaat toegestaan is. Die laaste oktet laat dan 'n maksimum van 254 stelsels op elke spesifieke Ethernet toe. Dit is 254 aangesien die gebruik van 0 en 255 nie toelaatbaar is nie. Vir stelsel waarvan die adres onbekend is word 0 gebruik en 255 word gebruik vir boodskappe ("Broadcast") wat na elke masjien op die netwerk gestuur moet word [16, p. 22].

Die doel van IP is die verskaffing van dienste aan 'n verskeidenheid van netwerk omgewings en daarom word daar voorsiening gemaak vir drie verskillende klasse van moontlike netwerkkonfigurasies [10, p. 37]. Hierdie klasse is dan as volg:

- A - Baie steunrekenaars op min netwerke
- B - Medium verspreiding van steunrekenaars en netwerke
- C - Min steunrekenaars op baie netwerke

Daar kan maklik onderskei word tussen die klasse indien gekyk word na die eerste veld in die adres [10, p. 39].

TABEL 3.1 Klasse van die IP adres

Waarde	Klas
1-127	A
128-191	B
192-223	C
224-255	D (Tans nie in gebruik)

Normaalweg word 'n naam aan 'n stelsel gegee in plaas van om die 32-bis adres te gebruik. Indien 'n naam wel gebruik word, sal die netwerksagteware in 'n databasis gaan kyk vir die regte 32-bis Internetadres [16, p. 18]. Hierdie diens word gelewer deur 'n DNS ("Domain Name Service") wat met behulp van die databasis 'n Naam-na-IP vertaling doen [10, p. 71].

TCP/IP maak gebruik van verbindinglose tegnologie [16, p. 7]. Inligting word as datagramme gestuur. Datagramme is 'n versameling van data wat as 'n enkele

boodskap gestuur word en elkeen word individueel oor die netwerk gestuur. Daar word ook voorsiening gemaak vir verbindings, met ander woorde 'gesprekke' of sessies wat langer as een datagram duur. Die inligting van so 'n sessie word nog steeds opgebreek, op een of ander vlak, in kleiner datagramme wat die netwerk as individuele datagramme hanteer.

Byvoorbeeld: Indien 'n 15000 oktet lêer gestuur moet word, sal die meeste netwerke dit nie kan hanteer nie [16, p. 7]. Die protokol sal dus die lêer opbreek in 500 oktetgrootte datagramme wat dan as 30 individuele datagramme hanteer sal word. Die opbreek van data in kleiner blokke, sal die hertransmissietyd van foutief ontvangde blokke data verkort. Die datagramme kan egter in enige volgorde ontvang word aangesien die netwerk nie tred hou met die beweging daarvan nie. Dit kan ook gebeur dat daar 'n fout in die netwerk mag voorkom en 'n datagram dan weer gestuur moet word. Die protokol by die ontvanger verseker egter dat die 15000 oktet lêer weer uit die individuele datagramme opgebou word.

L.W. Die term 'datagram' en 'pakkie' is baie dieselfde, maar by TCP/IP word die term datagram gebruik aangesien dit 'n eenheid van data is [16, p. 7]. 'n Pakkie is iets wat fisies in 'n Ethernet of 'n draad is. In baie gevalle bestaan dit eenvoudig uit 'n datagram, maar wanneer TCP/IP op 'n X25 netwerk gebruik word, word die datagramme in pakkies van 128 woorde opgebreek. Dit is egter onsigbaar vir IP en die pakkies word aan die ontvangkant weer aanmekaar gelas voordat die datagram deur TCP/IP geprosesseer word. Een IP datagram bestaan dus uit 'n klomp pakkies wat weer aanmekaar gelas word.

3.2.5 Die TCP vlak

Twee verskillende protokolle is verantwoordelik vir die hantering van TCP/IP datagramme, naamlik [16, p. 7]:

- TCP ("Transmission Control Protocol") is verantwoordelik vir die opbreek van data in datagramme, die samestelling van die datagramme nadat dit gestuur is, die hersending van datagramme wat verlore gegaan het en die herorganisering van die datagramme in die regte volgorde.
- IP ("Internet Protocol") is verantwoordelik vir die roetebeheer van die individuele datagramme.

Dit mag dalk voorkom asof TCP al die werk doen, wat wel waar is in klein netwerke, maar by Internet is dit 'n reuse taak om elke datagram by sy regte eindbestemming uit te bring. Een verbinding kan vereis dat die datagram deur verskeie netwerke gestuur word om sy bestemming te bereik. Om tred te hou met al die roetes na alle bestemmings en die aanpasbaarheid van stelsels tussen al die verskillende transportmedia, is 'n komplekse taak. Let op dat die koppelvlak tussen TCP en IP relatief eenvoudig is. TCP gee 'n datagram met bestemming aan IP. IP weet nie of die datagram deel vorm van ander data nie, maar sorg dat dit versend word.

Dit is duidelik dat dit nie voldoende is om slegs die datagram by sy regte bestemming te besorg nie, maar daar moet ook tred gehou word met die verskillende verbindinge in 'n gegewe stelsel. Die taak of proses waarna verwys word as de-multipleksing, word deur TCP gebruik om te bepaal van watter verbinding die datagram deel is. Daar is egter verskillende vlakke van de-multipleksing wat deur TCP/IP gebruik word. Die inligting wat benodig word vir die de-multipleksing word vervat in 'n reeks van kopetikette. 'n Kopetiket

is slegs 'n paar oktette wat deur een of ander protokol aan die begin van 'n datagram gelas word, om tred te hou met die datagram. 'n Eenvoudige voorbeeld hiervan is die plasing van 'n brief in 'n koevert en die adressering daarvan. Met moderne netwerke word die proses 'n paar maal herhaal.

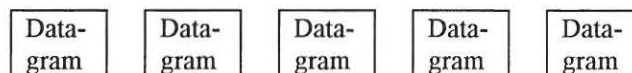
Die volgende is 'n oorsig van die kopetikette wat deur 'n TCP/IP netwerk aan 'n datagram gelas word:

By die oorsending van 'n datastroom, soos byvoorbeeld 'n lêer, na 'n ander rekenaar lyk die data as volg:



FIGUUR 3.3 Voorstelling van 'n datastroom

TCP breek die data op in hanteerbare stukke en volgens die grootte van die datagramme wat die spesifieke netwerk kan hanteer. In der waarheid kommunikeer TCP aan beide kante met mekaar, sodat die datagramgrootte wat hanteer kan word, bekend is en die kleinste grootte deur albei kante gekies kan word. Die data lyk dus nou as volg:



FIGUUR 3.4 Opbreking van datastroom in datagramme

TCP sit dan 'n kopetiket voor aan elke datagram, wat gewoonlik bestaan uit tenminste 20 oktette, waarvan die belangrikstes die bron, die bestemming, die poortnommer en die sekvensienommer is. Die poortnommers word gebruik om tred te hou met die verskillende gesprekke. Indien drie verskillende persone gelyktydig lêers oordra, sal TCP byvoorbeeld poort nommers 1000, 1001, en 1002 aan die oordragte toeken. Wanneer 'n gebruiker 'n datagram stuur, word

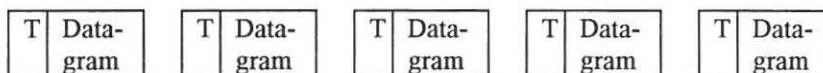
dit die bronpoortnommer, aangesien die gebruiker die bron van die datagram is. Die ontvanger TCP ken ook 'n poortnommer, wat aan beide kante bekend moet wees, aan die sessie toe. By die aanvang van die gesprek, word hierdie inligting uitgeruil en in die bestemmingspoortveld opgeneem. By die terugsending van die datagram word die bestemming- en bronvelde omgeruil. Elke datagram het 'n sekwensienommer. Dit word gebruik om te verseker dat die ander TCP die datagramme in die regte volgorde ontvang en dat 'n datagram nie verlore gaan nie. TCP nommer die oktette en nie die datagramme nie. Dit wil sê indien daar 500 oktette in elke datagram is, sal die eerste datagram 0 genummer word, die tweede 500, die volgende 1000, 1500 ensovoorts. Die kontrolesom word gebruik om te bepaal of daar foute in die oordrag voorgekom het. Dit word bepaal deur al die oktette in die datagram op te tel. Die antwoord word in die kopetiket gesit en die ontvang-TCP bepaal dan weer die kontrolesom en vergelyk die waardes. Indien die waardes verskil, het daar 'n fout voorgekom met die transmissie en moet die datagram weer hersend word.

Die datagram lyk dus nou as volg:

Bronpoort				Bestemmingspoort				
Sekwensienommer								
Bevestigingsnommer								
Data verpla- sing	Gereser- veerd	U R G	A C K	P S H	R S T	S Y N	F I N	Venster
Kontrolesom				Dringendheidswyser				
Opsies								
Data								

FIGUUR 3.5 TCP kopetiket

Indien die TCP kopetiket afkort as "T", sal die lêer as volg lyk:



FIGUUR 3.6 Samevoeging van TCP kopetiket vooraan elke datagram

Die volgende items in die kopetiket word gebruik vir die beheer van die verbinding:

- *Bevestiging van dataontvangs deur ontvanger.* Hierdie bevestiging word gedoen met behulp van 'n datagram, waarvan die bevestigingsveld ingevul

is, deur die ontvanger. Indien die sender 'n datagram met 'n waarde van 1500 in die bevestigingsveld ontvang, is dit 'n aanduiding dat alle data tot en met oktet nommer 1500 foutloos ontvang is. Indien die sender nie 'n bevestiging binne 'n redelike tydperk ontvang nie, sal die data weer gestuur word.

- *Verkryging van die waarde in die vensterveld.* Die venster word gebruik om die hoeveelheid data wat op een slag gestuur kan word, te beheer. Dit is egter nie prakties om na die versending van elke datagram te wag vir 'n bevestiging voordat die volgende datagram gestuur word nie, aangesien dit die oordrag sal vertraag. Indien die sender vinniger is as die ontvanger, kan die data vinniger geskryf word as wat die ontvanger dit kan verwerk. Sodoende sal data verlore gaan. Beide kante van die verbinding toon aan hoeveel data op 'n sekere moment verwerk kan word, deur die aantal oktette in die vensterveld aan te toon. Die waarde in die vensterveld verminder met ontvangs van data en indien die waarde tot zero daal, moet transmissie deur die sender gestaak word. Die vensterwaarde word weer vergroot namate data deur die ontvanger geprosesseer word.

3.2.6 Die IP vlak

Met die deursending van datagramme deur TCP aan IP moet die Internetadres van die ontvanger rekenaar deur eersgenoemde verstrek word. IP benodig hierdie inligting, aangesien dit slegs die roetebeheer van die datagram moet doen [10, p. 23]. Om deurgange in staat te stel om die datagram aan te stuur, moet IP sy eie kopetiket voor aan die datagram voeg. Die hoofdele van die kopetiket is die bron, die bestemmings adres, die protokolnommer en 'n kontrolesom. Die bron adres is die adres van die masjien waarop die gebruiker werk, wat verseker dat die ontvanger weet waarvandaan die data gestuur word.

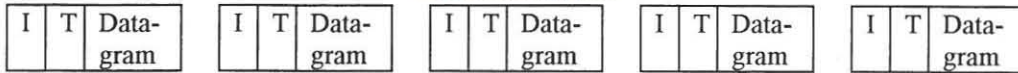
Die bestemmingsadres is die adres van die ontvanger masjien. Die protokol nommer bepaal dat die ontvangs IP die datagram aan TCP sal stuur. Weens die feit dat nie net TCP nie, maar ook ander protokolle IP kan gebruik, is dit belangrik om hierdie veld in te vul. Die kontrolesom stel IP in staat om te bepaal of daar foute in die transmissie voorgekom het. Dit moet in gedagte gehou word dat TCP en IP, in hul onderskeie kopetikette, verskillende kontrolesomme gebruik.

Die IP kopetiket kan dus as volg voorgestel word:

Versie	IHL	Tipe diens	Totale lengte	
Identifikasie			Mer- kers	Fragmentverplasing
Lewensduur	Protokol		Kopetiket kontrolesom	
Bronadres				
Bestemmingsadres				
Opsies			Buffer	
Data				

FIGUUR 3.7 IP kopetiket

Indien die IP kopetiket as "I" en die TCP kopetiket as "T" afgekort word, word die datagram as volg voorgestel:



FIGUUR 3.8 Samevoeging van IP en TCP kopetikette, sowel as datagramme

3.2.7 Koppelpunte bekend aan TCP netwerk

Tot dusver is die volgende bespreek:

- Die opbreking van 'n datastroom in 'n aantal datagramme.
- Die versending van data na 'n ontvanger rekenaar.
- Die rangskikking van datagramme in die regte volgorde.

Ten einde effektief te wees, is die volgende egter ook noodsaaklik [16, p. 12]:

- Die bewerkstelling van die verbinding na 'n spesifieke rekenaar.
- Aantekening op die rekenaar.
- Die bepaling van verlangde lêer en ook die beheer van die transmissie van die lêer.
- Bepaling van verlangde protokol.

Indien 'n ander toepassing as lêeroordrag benodig word, word 'n ander protokol vereis [16, p. 12]. Hierdie protokolle word toepassingsprotokolle genoem wat in samewerking met, of bo-op TCP/IP gebruik word. TCP en IP hanteer die vloei van data en die toepassingsprotokolle hanteer die data wat gestuur word, as een datastroom. TCP en IP hanteer dus alle netwerk funksies en verseker dat die data foutloos ontvang word.

Voordat 'n lêer na 'n bekende Internet adres gestuur kan word, moet daar eers 'n verbinding met die FTP bediener gemaak word [16, 13]. Aangesien netwerksagteware in die algemeen gespesialiseerd is vir spesifieke take, benodig die meeste stelsels aparte programme vir die hantering van lêeroordrag, afstandige aantekenings, elektroniese pos, ensovoorts. Wanneer die verbinding na die ander rekenaar gemaak word, moet daar dus gespesifiseer word dat dit 'n FTP koppeling is. Dit word moontlik gemaak deur gebruikmaking van sogenaamde welbekende koppelpunte vir elke bediener. Soos voorheen genoem, maak TCP gebruik van poorte om tred te hou met individuele netwerksessies. Gebruikersprogramme gebruik meestal willekeurige poortnommers, terwyl spesifieke poortnommers aan sagteware toegeken word wat wag op die aanvang van sessies. By die versending van lêers word FTP sagteware gebruik. Die sessie sal begin word met 'n willekeurige poortnommer aan die kant van die bediener, terwyl poort nommer 21 vir die ander gespesifiseer sal word. Hierdie poort 21 staan bekend as 'n welbekende koppelpunt vir FTP. 'n Verkorte lys van die mees algemene welbekende koppelpunte bekend aan 'n TCP netwerk, kan in Tabel 3.2 gesien word.

TABEL 3.2 Welbekende koppelpunte in TCP

Desimale waarde	Naam	Beskrywing
7	ECHO	"Echo"
9	DISCARD	"Discard"
20	FTP-data	"File Transfer (Data)"
21	FTP	"File Transfer Protocol (Control)"
23	TELNET	"Telnet"
25	SMTP	"Simple Mail Transfer Protocol"
79	FINGER	"Finger"

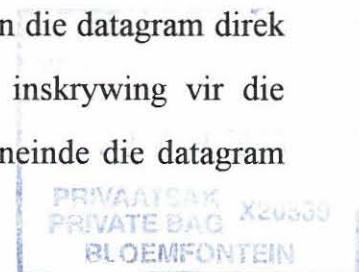
3.2.8 Roetebepaling van die TCP netwerk

Uit die voorafgaande paragrawe is dit duidelik dat IP verantwoordelik is dat die datagram die bestemmingsadres bereik. Hierdie proses word roetebepaling genoem en die manier waarop dit uitgevoer word, word deur die spesifieke implementasie bepaal [16, p. 20].

IP veronderstel dat die stelsel aan een of ander plaaslike netwerk gekoppel is en aanvaar dat datagramme na enige ander stelsel, wat aan hierdie netwerk gekoppel is, gestuur kan word. Wanneer 'n datagram na 'n ander netwerk gestuur moet word, moet daar gebruik gemaak word van 'n deurgang om die twee netwerke aan mekaar te koppel. 'n Deurgang is 'n stelsel wat een netwerk met een of meer ander netwerke verbind. 'n Gewone rekenaar wat aan meer as een netwerk gekoppel is, kan as 'n deurgang gebruik word, indien die sagteware op hierdie masjien so opgestel is dat enige datagramme, wat bedoel is vir 'n ander netwerk, na daardie spesifieke netwerk oorgedra sal word.

Roetebepaling in IP word uitsluitlik deur die netwerknommer in die bestemmingsadres gedoen [16, p. 20]. Elke rekenaar beskik oor 'n tabel met netwerknommers, waaraan elkeen 'n deurgang toegeken is, wat gebruik moet word om daardie netwerk te bereik. Dit is egter nie noodsaaklik vir die deurgang om direk verbind te wees met die netwerk nie, maar dit is slegs die beste roete om die netwerk te bereik.

By die versending van 'n datagram word eers bepaal of die bestemmingsadres op die rekenaar se eie netwerk is. Indien dit die geval is, kan die datagram direk gestuur word, maar indien nie, verwag die rekenaar 'n inskrywing vir die netwerk waaraan die bestemmingsrekenaar gekoppel is, teneinde die datagram na die deurgang te stuur.



97/4523

3.2.9 Datagramfragmentasie en hersamestelling

TCP/IP is ontwikkel vir die gebruik saam met 'n wye verskeidenheid tipes netwerke. Netwerkontwerpers het egter nie eenstemmigheid oor die grootte van die pakkies wat gebruik moet word nie. Ethernet pakkies kan 1500 oktette lank wees, terwyl Arpanet pakkies 'n maksimum lengte van ongeveer 1000 oktette het [16, p. 23]. Sommige van die vinniger netwerke gebruik 'n baie groter pakkiegrootte. Indien IP die kleinste moontlike grootte gebruik, sal dit veroorsaak dat die netwerk baie stadig is, aangesien daar aan elke datagram kopetikette gelas word. In die geval van 'n baie klein pakkie bestaan dit bykans net uit kopetikette en baie min werklike data word gestuur. Wanneer groot lêers gestuur word, is dit dus beter om groot pakkies te gebruik. Alhoewel die versending van die grootste moontlike pakkies verkieslik is, moet daar ook voorsiening gemaak word vir netwerke wat net klein pakkies kan hanteer. By die keuse van die grootte van die pakkies wat gebruik moet word, is die volgende van belang: Beide rekenaars stuur die maksimum datagramgrootte wat hulle kan hanteer. Die kleinste van die twee word vir die res van die sessie gebruik. Hierdie metode stel dus netwerke, wat groot datagramme kan hanteer, in staat om dit te gebruik terwyl dit ook moontlik is om met netwerke te kommunikeer wat nie die groot datagramme kan hanteer nie.

Wanneer daar nie met sekerheid bepaal kan word of die volle roete die datagramgrootte sal kan hanteer nie, word 576-greep datagramme algemeen gebruik, aangesien alle toepassings hierdie grootte moet ondersteun [16, p. 24].

3.3 Ander protokolle as TCP: UDP en ICMP

TCP is verantwoordelik vir die opbreek van data in kleiner datagramme, die hersamestelling van hierdie datagramme in die regte volgorde en ook vir foutkorreksie. Daar bestaan egter baie toepassings waar die blok data wat gestuur moet word klein genoeg sal wees om in een datagram te pas. Aangesien die ingewikkelde funksies van TCP nie in hierdie gevalle benodig word nie, bestaan daar ander moontlikhede.

Die mees algemene alternatief is UDP ("User Datagram Protocol") [16, p. 17]. UDP is ontwikkel vir toepassings waar dit onnodig is om sekwenisiële datagramme weer saam te stel, met ander woorde die data sal alles in een datagram gestuur kan word. UDP word basies net soos TCP in die stelsel geïntegreer. Daar is ook 'n UDP kopetiket wat deur die sagteware voor die data aangelas word, net soos die geval met die TCP kopetiket. UDP stuur die data na IP, waar die IP kopetiket voor aangelas word, asook die invoeging van die UDP protokolnommer in die protokolveld. UDP kan nie dieselfde hoeveelheid funksies as TCP verrig nie en voorsien slegs poortnommers vir die gebruik deur verskillende pakette. UDP, net soos TCP, maak gebruik van welbekende poortnommers vir die gebruik deur bedieners wat UDP gebruik. Die UDP kopetiket is kleiner as die van TCP. Benewens die bestemming, bron poortnommers en 'n kontrolesom, is daar geen verdere ooreenkoms tussen UDP en TCP nie.

'n Ander alternatief vir TCP is ICMP ("Internet Control Message Protocol"). ICMP word gebruik vir die versending van foutboodskappe en boodskappe wat bedoel is vir ander TCP/IP sagteware, in plaas van 'n spesifieke gebruikersprogram. ICMP kan ook gebruik word om inligting te bekom van 'n ander netwerk. Dit is dieselfde as UDP in die opsig dat die data ook in een

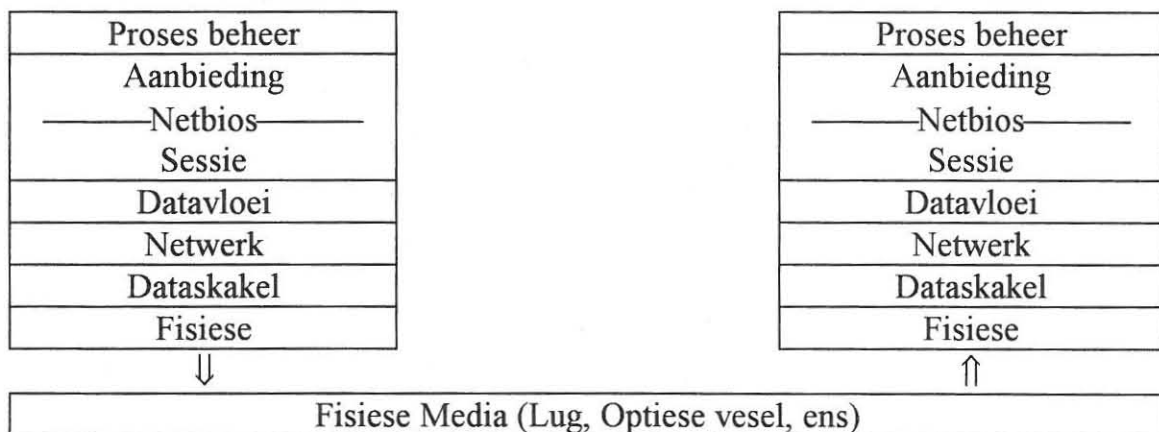
datagram pas, maar dit is nog eenvoudiger as UDP. Die kopetiket vir ICMP bevat geen poort nommers nie, aangesien alle ICMP navrae deur die netwerksagteware self hanteer word.

3.4 NetBIOS as netwerkprotokol

"Network Basic Input/Output System" (NetBIOS) is 'n toepassings-programmeringskoppelvlak vir dataoordrag tussen databronne en dataontvangers. Toepassingsagteware moet gebruik maak van NetBIOS se dienste deur 'n spesifieke opdragorde. Normaalweg sal dataoordrag deur middel van NetBIOS tussen twee rekenaars, wat aan 'n lokale areanetwerk (LAN) gekoppel is, plaasvind. Dit sal egter moontlik wees vir twee toepassings, op een rekenaar, om van NetBIOS gebruik te maak vir die oordrag van data [39, p. 3].

3.4.1 NetBIOS in die netwerkhierargie

Met verwysing na die OSI Verwysingsmodel kan in Figuur 3.9 gesien word, waar NetBIOS in die netwerkhierargie pas [39, p. 3].



FIGUUR 3.9 NetBIOS in die netwerkhierargie

NetBIOS is ook oorweeg as 'n moontlike protokol, maar aangesien TCP/IP beter voorsiening maak vir seriale kommunikasie, is daar besluit om TCP/IP en nie NetBIOS te gebruik nie.

3.5 Opsomming

Hierdie hoofstuk het gehandel oor die keuse wat gemaak is ten opsigte van die gebruik van 'n geskikte protokol en ook 'n volledige beskrywing van die gekose protokol.

TCP/IP is gekies as protokol, aangesien hierdie protokol voorsiening maak vir die erkenning van foute wat gedurende transmissie voorgekom het, deur gebruik te maak van kontrolebisse. TCP/IP maak ook voorsiening vir die hersending van datagramme waar daar foute voorgekom het.

Koppeling met die seriepoort is ook maklik wanneer TCP/IP gebruik word, aangesien daar van SLIP drywers gebruik gemaak kan word.

Wanneer al die voordele wat verkry word, indien daar van TCP/IP gebruik gemaak word, in ag geneem word, is dit duidelik dat TCP/IP 'n goeie keuse vir 'n netwerkprotokol is, vir gebruik in mobiele mediese eenhede.

HOOFSTUK 4

SAGTEWARE ONTWIKKELING

Hierdie hoofstuk beskryf die implementering en integrering van die TCP/IP en sellulêre koppelvlakke in die sagteware.

4.1 Algemene beskrywing van die sagteware

Alle sagteware wat vir die projek gebruik is, is in C++ geskryf. C++ is gekies omdat dit 'n programmeringstaal is wat baie algemeen gebruik word en ook baie kragtig is.

Die SLIP drywers wat benodig word deur die sagteware, is baie algemeen beskikbaar op Internet en daar is dus geen rede om hierdie sagteware oor te skryf nie. Hierdie drywers is gebruik by die evaluering van die sagteware.

4.1.1 Ontwikkeling van die netwerkontvanger sagteware

Eerstens is daar met die ontwikkeling van die sagteware vir die ontvangrekenaar begin. Hierdie sagteware moet oor die vermoë beskik om afstandige aantekening toe te laat, sowel as die stoor van die data wat ontvang word, in 'n toepaslike lêer. Die sagteware het dit ten doel om voorbereid te wees op die koppeling van 'n netwerksender en om, onder die beheer van die sender, onmiddelik te begin met ontvangs van datapakette. 'n Wagwoord kan as beskerming teen ongemagtigde gebruik van data gebruik word. Daar is eerstens

begin met die skryf van 'n mees basiese program wat eers net die afstandige aantekening deur middel van 'n TCP/IP koppeling sal toelaat.

Figuur 4.1 toon die vloeddiagram van die volledige ontvangersagteware. Die modemprosedures word in Paragraaf 4.1.3 aangespreek.

Die sagteware (server.c) wat vir die doel ontwikkel is, kan in Bylaag 4 gesien word. Die prosedures wat in die sagteware gebruik is, is as volg:

- ◆ **unsigned ServNetInit(unsigned Index);**

Maak die TCP-koppeling oop met behulp van die subroetine TCPOpen().

- ◆ **enum flag ServConnect(unsigned Index);**

Hierdie roetine bepaal of die koppeling suksesvol gemaak is en gee 'n toepaslike boodskap op die skerm.

- ◆ **unsigned TxBuffer(unsigned Index, char *ptrTxBuf, unsigned TxLen);**

Roep die SetLenData() roetine met die toepaslike parameters.

- ◆ **unsigned TxdBuffer(unsigned Index);**

Roep die TxLenData() roetine met die toepaslike parameters en toets vir foutboodskappe.

- ◆ **unsigned RxBuffer(unsigned Index, char *ptrRxBuf);**

Roep die SetLenData() roetine.

- ◆ **unsigned RxdBuffer(unsigned Index, unsigned *ptrRxLen);**

Roep die RxLenData() roetine met die toepaslike parameters en toets vir foutboodskappe.

- ◆ **unsigned TCPRd(unsigned ofsTCB, struct trLen *pachBuf, unsigned *pcBuf);**

Generering van 'n onderbreking vir die lees van data vanaf die drywers. Inligting soos lêernaam ensovoorts word terselfdertyd verkry met die gebruikmaking van die trLen struktuur. Die volgende registers word as volg gebruik:

Register (Begin):	AH:	TCP_READ
	BX:	TCB adres
	CX:	Telling
	DX:SI	Buffer adres (SEG:OFF)
Register (Uit):	CX:	Ware lees hoeveelheid
	AX:	Fout

- ◆ **unsigned TCPRd1(unsigned ofsTCB, char *pachBuf, unsigned *pcBuf);**

Generering van onderbreking vir die lees van data sonder verdere inligting soos verkry deur TCPRd(). Die volgende registers word as volg gebruik:

Register (Begin):	AH:	TCP_READ
	BX:	TCB adres

	CX:	Telling
	DX:SI	Buffer adres (SEG:OFF)
Register (Uit):	CX:	Ware lees hoeveelheid
	AX:	Fout

◆ **unsigned TCPWr(unsigned ofsTCB, struct trLen far *pachBuf, unsigned *pcBuf);**

Generering van 'n onderbreking vir die skryf van data na die drywers. Inligting soos lêernaam ensovoorts word terselfdertyd gestuur met die gebruikmaking van die trLen struktuur. Die volgende registers word as volg gebruik:

Register (Begin):	AH:	TCP_WRITE
	AL:	1 vir dringende data
	BX:	TCB adres
	CX:	Telling
	DX:SI	Buffer adres (SEG:OFF)
Register (Uit):	CX:	Ware skryf hoeveelheid
	AX:	Fout

◆ **unsigned TCPWr1(unsigned ofsTCB, char *pachBuf, unsigned *pcBuf);**

Generering van onderbreking vir die skryf van data sonder verdere inligting soos gestuur deur TCPWr(). Die volgende registers word as volg gebruik:

Register (Begin):	AH:	TCP_WRITE
	AL:	1 vir dringende data
	BX:	TCB adres
	CX:	Telling
	DX:SI	Buffer adres (SEG:OFF)
Register (Uit):	CX:	Ware skryf hoeveelheid

AX: Fout

- ◆ **unsigned int TCPOpen(TCP_OP_P op, unsigned far *psegTCB, unsigned far *pofsTCB);**

Generering van onderbreking vir die maak van 'n TCP-koppeling. Die volgende registers word as volg gebruik:

Register (Begin): AH: TCP_OPEN
BX: CX: Parameters adres (SEG:OFF)

Register (Uit): BX: CX: TCB adres (SEG:OFF)
AX: Fout

- ◆ **unsigned long int GetIPAddr(char szAddr[]);**

Verkry die IP adres vanaf die naam van die rekenaar soos verkry vanaf die konfigurasie lêer. Die volgende registers word as volg gebruik:

Register (Begin): AH: GETIPADR
BX: CX: Aanwyser ("Pointer") na naam

Register (Uit): BX: CX: IP adres (0 indien nie gekry)

- ◆ **void APIAlive(void);**

Stuur van data sodat netwerkverbinding nie verbreek word indien data nie gestuur word nie.

- ◆ **void TxLenData (unsigned segTCP, unsigned ofsTCP, struct TrXfer *ptrNCB);**

Bepaal of string data klaar gestuur is. Indien dit nie klaar gestuur is nie, sal die volgende blok data gestuur word.

- ◆ **void SetLenData (struct TrXfer *ptrNCB, char *ptradbBuf, unsigned cSize);**

Bepaal die lengte van data wat gestuur moet word.

- ◆ **void RxLenData (unsigned ofsTCP, struct TrXfer *ptrNCB);**

Bepaal of data klaar ontvang is en wat die lengte van die blok is wat ontvang is.

- ◆ **void Write_Buf(void);**

Skryf die buffer se inhoud na 'n lêer.

- ◆ **void Open_File(void);**

Maak die bestemmingslêer op die ontvangsrekenaar oop.

- ◆ **void TimeSlice(void);**

Roetine vir die beheer van die sagteware. Die bepaling van die volgorde van uitvoering van die subroetines.

◆ **void Init(void);**

Gee toepaslike foutboodskap indien nodig wanneer 'n geldige netwerkkoppeling nie bestaan nie.

◆ **void User(void);**

Die verkryging van netwerkname vanaf die konfigurasielêer.

◆ **void Menu(void);**

Die uitleg van die skerm word deur hierdie roetine behartig.

◆ **void close_tcp(unsigned tcb);**

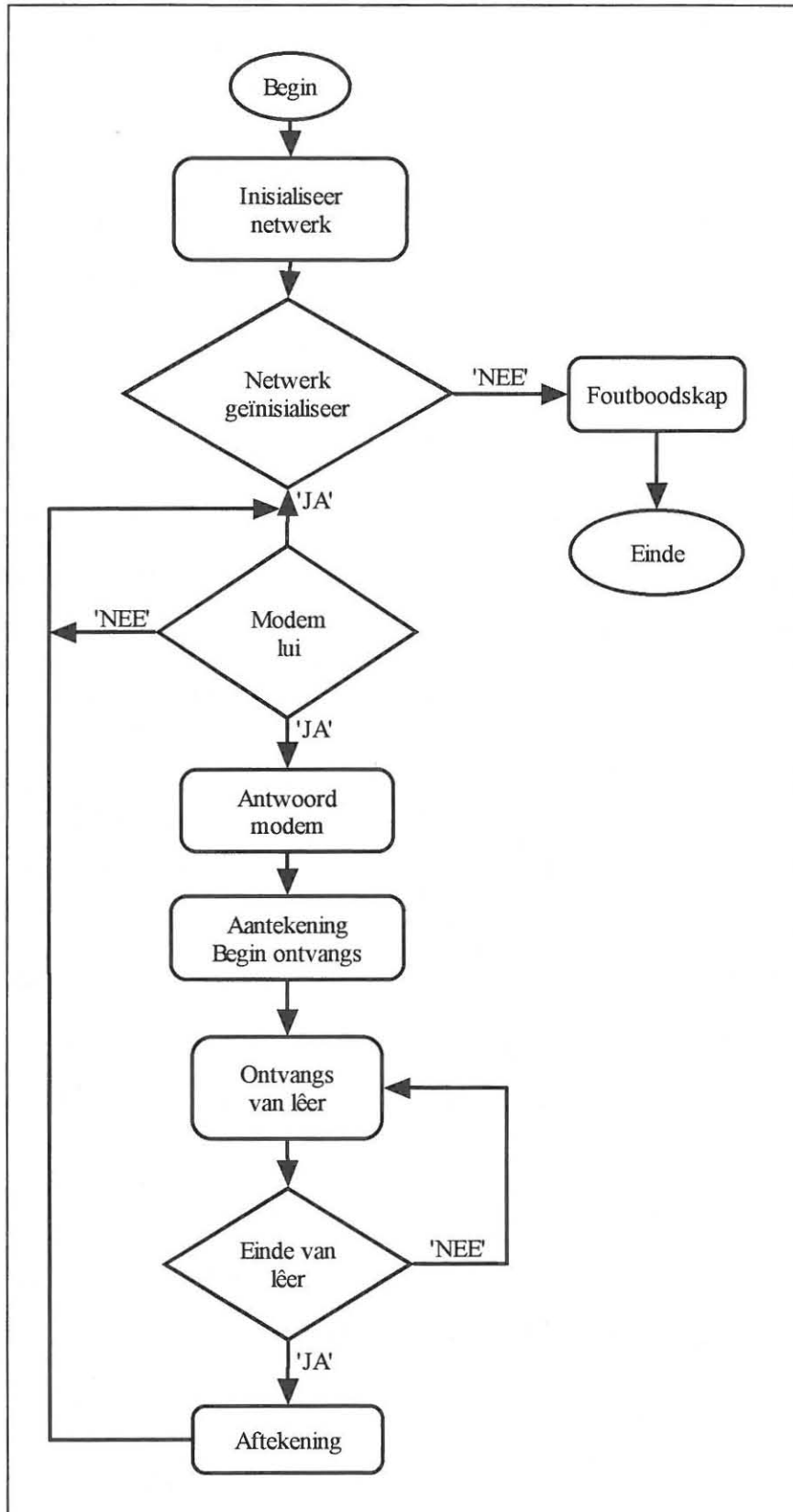
Maak die TCP verbinding toe. Die volgende registers word as volg gebruik:

Register (Begin):	AH:	TCP_CLOSE
	BX:	TCB adres (OFF)
Register (Uit):	AX:	Fout

◆ **void del_tcp(unsigned tcb);**

Maak die TCB van die verbinding skoon. Die volgende registers word as volg gebruik:

Register (Begin):	AH:	TCP_DELET
	BX:	TCB adres (OFF)
Register (Uit):	AX:	Fout



FIGUUR 4.1 Vleiediagram van ontvanger sagteware

4.1.2 Ontwikkeling van sagteware vir die netwerksender

Die sagteware vir die afstandige rekenaar moet oor die vermoë beskik om 'n afstandige aantekening te doen, 'n lêer te selekteer en te versend. Albei rekenaars moet ook 'n modemkoppelvlak, hetsy sellulêr- of 'n gewone telefoon, ondersteun. Die afstandige rekenaarsagteware moet dus ook die skakeling van die regte telefoonnommer behartig, om sodoende 'n sessie te kan begin (Sien Paragraaf 4.1.3 vir beskrywing van modem sagteware). Hierdie sagteware is geskryf tot op die stadium waar 'n afstandige aantekening moontlik sou wees, indien die rekenaars deur middel van 'n nulmodemkabel aan hul RS232 poorte gekoppel is. Twee rekenaars is dan ook so gekoppel en die sagteware is getoets en verander totdat die aantekening moontlik is.

Die ontwikkeling van die sendersagteware is meer gerig op die vervanging van 'n operateur. Data, soos byvoorbeeld X-straalbeelde, word deur middel van 'n skandeerder ingelees en op 'n sekere indeks gestoor. Die sendersagteware monitor die skyf en indien nuwe data beskikbaar is, word onmiddelik begin met die koppeling na die ontvanger. As die fisiese verbinding bestaan, word die netwerkkoppeling geopen waarna dataoordrag begin.

Lêerhantering vorm 'n baie belangrike deel van die sagteware wat ontwikkel word en die volgende stap was dus om die sagteware aan te pas om eers 'n kort boodskap oor te stuur na die steunrekenaar. Nadat hierdie stap voltooi is, is die sagteware op die steunrekenaar sowel as die sagteware op die afstandige rekenaar aangepas om lêerversending moontlik te maak. Hier moet goeie beheer oor die datavloei uitgeoefen word, aangesien die meeste lêers groter as een datagram is. Om dus een lêer te versend, word 'n paar datagramme benodig. Die lêer word dus opgebreek, versend en weer aan die ontvangerkant saamgestel.

'n Protokol vir dataoordrag is ontwikkel wat as volg werk: Daar is besluit om 'n verdere kopetiket in te voeg voor aan elke datagram. Hierdie kopetiket lyk as volg:

TABEL 4.1 Kopetiket ingevoeg om datavloei te vergemaklik

Beskrywing	Waarvoor gebruik
char zID[15]	"BytesToExpect" (Word gebruik vir datagram herkenning)
unsigned cLen	Grootte van die datagram
unsigned J Value	Aantal woorde gelees in lêer
char File Name[13]	Lêernaam
unsigned Command	0 = Skryf data na lêer 1 = Maak lêer oop 2 = Maak lêer toe

Hoewel die kopetiket veroorsaak dat minder data in elke datagram gestuur word, is die grootte van die kopetiket so klein dat dit nie 'n werklike verskil op die oordragspoed van die data maak nie.

'n Vloedidiagram van die sender sagteware kan in Figuur 4.2 gesien word en die subroetines wat vir die sagteware (host.c) gebruik is, is as volg (Sien Bylaag 2):

♦ **unsigned TCPRd(unsigned ofsTCB, struct trLen *pachBuf, unsigned *pcBuf);**

Generering van 'n onderbreking vir die lees van data vanaf die drywers. Inligting soos lêernaam ensovoorts word terselfdertyd verkry met die gebruikmaking van die trLen struktuur. Die volgende registers word as volg gebruik:

Register (Begin): AH: TCP_READ

	BX:	TCB adres
	CX:	Telling
	DX:SI	Buffer adres (SEG:OFF)
Register (Uit):	CX:	Ware lees hoeveelheid
	AX:	Fout

◆ **unsigned TCPRd1(unsigned ofsTCB, char *pachBuf, unsigned *pcBuf);**

Generering van onderbreking vir die lees van data sonder verdere inligting soos verkry deur TCPRd(). Die volgende registers word as volg gebruik:

Register (Begin):	AH:	TCP_READ
	BX:	TCB adres
	CX:	Telling
	DX:SI	Buffer adres (SEG:OFF)
Register (Uit):	CX:	Ware lees hoeveelheid
	AX:	Fout

◆ **unsigned TCPWr(unsigned ofsTCB, struct trLen far *pachBuf, unsigned *pcBuf);**

Generering van 'n onderbreking vir die skryf van data na die drywers. Inligting soos lêernaam ensovoorts word terselfdertyd gestuur met die gebruikmaking van die trLen struktuur. Die volgende registers word as volg gebruik:

Register (Begin):	AH:	TCP_WRITE
	AL:	1 vir dringende data
	BX:	TCB adres
	CX:	Telling
	DX:SI	Buffer adres (SEG:OFF)
Register (Uit):	CX:	Ware skryf hoeveelheid

AX: Fout

◆ **unsigned TCPWr1(unsigned ofsTCB, char *pachBuf, unsigned *pcBuf);**

Generering van onderbreking vir die skryf van data sonder verdere inligting soos gestuur deur TCPWr(). Die volgende registers word as volg gebruik:

Register (Begin):	AH:	TCP_WRITE
	AL:	1 vir dringende data
	BX:	TCB adres
	CX:	Telling
	DX:SI	Buffer adres (SEG:OFF)
Register (Uit):	CX:	Ware skryf hoeveelheid
	AX:	Fout

◆ **unsigned int TCPOpen(TCP_OP_P op, unsigned far *psegTCB, unsigned far *pofsTCB);**

Generering van onderbreking vir die maak van 'n TCP-koppeling. Die volgende registers word as volg gebruik:

Register (Begin):	AH:	TCP_OPEN
	BX:CX:	Parameters adres (SEG:OFF)
Register (Uit):	BX:CX:	TCB adres (SEG:OFF)
	AX:	Fout

◆ **unsigned long int GetIPAddr(char szAddr[]);**

Verkry die IP adres vanaf die naam van die rekenaar soos verkry vanaf die konfigurasie lêer. Die volgende registers word as volg gebruik:

Register (Begin):	AH:	GETIPADR
-------------------	-----	----------

BX: CX: Aanwyser ("Pointer") na naam
Register (Uit): BX: CX: IP adres (0 indien nie gekry)

◆ **void APIAlive(void);**

Stuur van data sodat netwerkverbinding nie verbreek word indien data nie gestuur word nie.

◆ **void TxLenData (unsigned segTCP, unsigned ofsTCP, struct TrXfer *ptrNCB);**

Bepaal of string data klaar gestuur is. Indien dit nie klaar gestuur is nie, sal die volgende blok data gestuur word.

◆ **void SetLenData (struct TrXfer *ptrNCB, char *ptradbBuf, unsigned cSize);**

Bepaal die lengte van data wat gestuur moet word.

◆ **void RxLenData (unsigned ofsTCP, struct TrXfer *ptrNCB);**

Bepaal of data klaar ontvang is en wat die lengte van die blok is wat ontvang is.

◆ **unsigned int NetConnect(char Host[], unsigned Index);**

Hierdie roetine stel die parameters van die TCB op en roep die TCPOpen roetine om die TCP koppeling te maak.

◆ **unsigned int NetConnected(unsigned int Index);**

Bepaal wat die status van die netwerk is, met ander woorde of 'n suksesvolle netwerkkoppeling bewerkstellig is.

◆ **unsigned TxBuffer(unsigned Index, char *ptrTxBuf, unsigned TxLen);**

Roep die SetLenData() roetine met die toepaslike parameters.

◆ **unsigned TxdBuffer(unsigned Index);**

Roep die TxLenData() roetine met die toepaslike parameters en toets vir foutboodskappe.

◆ **unsigned RxBuffer(unsigned Index, char *ptrRxBuf);**

Roep die SetLenData() roetine.

◆ **unsigned RxdBuffer(unsigned Index, unsigned *ptrRxLen);**

Roep die RxLenData() roetine met die toepaslike parameters en toets vir foutboodskappe.

◆ **int open_tcp(TCP_OP_P op);**

Generering van onderbreking vir die maak van 'n TCP-koppeling. Die volgende registers word as volg gebruik:

Register (Begin):	AH:	TCP_OPEN
	BX:CX:	Parameters adres (SEG:OFF)

Register (Uit): BX: CX: TCB adres (SEG: OFF)
 AX: Fout

◆ **void TimeSlice(void);**

Roetine vir die beheer van die sagteware. Die bepaling van die volgorde van uitvoering van die subroetines.

◆ **void WarningSound(void);**

Die opwekking van 'n waarskuwingsein deur middel van die rekenaar luidspreker wanneer 'n fout met die uitvoering van die sagteware ondervind word.

◆ **void User(void);**

Die verkryging van netwerkname vanaf die konfigurasielêer.

◆ **void NetInit(void);**

Gee toepaslike foutboodskap indien nodig wanneer 'n geldige netwerkkoppeling nie bestaan nie.

◆ **void Menu(void);**

Die uitleg van die skerm word deur hierdie roetine behartig.

◆ **void API_Loaded(void);**

Bepaal of die netwerkdrywers gelaai is.

◆ **void close_tcp(unsigned tcb);**

Maak die TCP verbinding toe. Die volgende registers word as volg gebruik:

Register (Begin):	AH:	TCP_CLOSE
	BX:	TCB adres (OFF)
Register (Uit):	AX:	Fout

◆ **void del_tcp(unsigned tcb);**

Maak die TCB van die verbinding skoon. Die volgende registers word as volg gebruik:

Register (Begin):	AH:	TCP_DELETE
	BX:	TCB adres (OFF)
Register (Uit):	AX:	Fout

◆ **char *err(unsigned errcode);**

Roetine om die numeriese fout om te skakel in toepaslike foutboodskap.

◆ **void open_window(char x1, char y1, char x2, char y2);**

Stoor die grafiese inligting voordat 'n venster bo-op 'n ander oopgemaak word.

◆ **void close_window(char x1,char y1,char x2,char y2);**

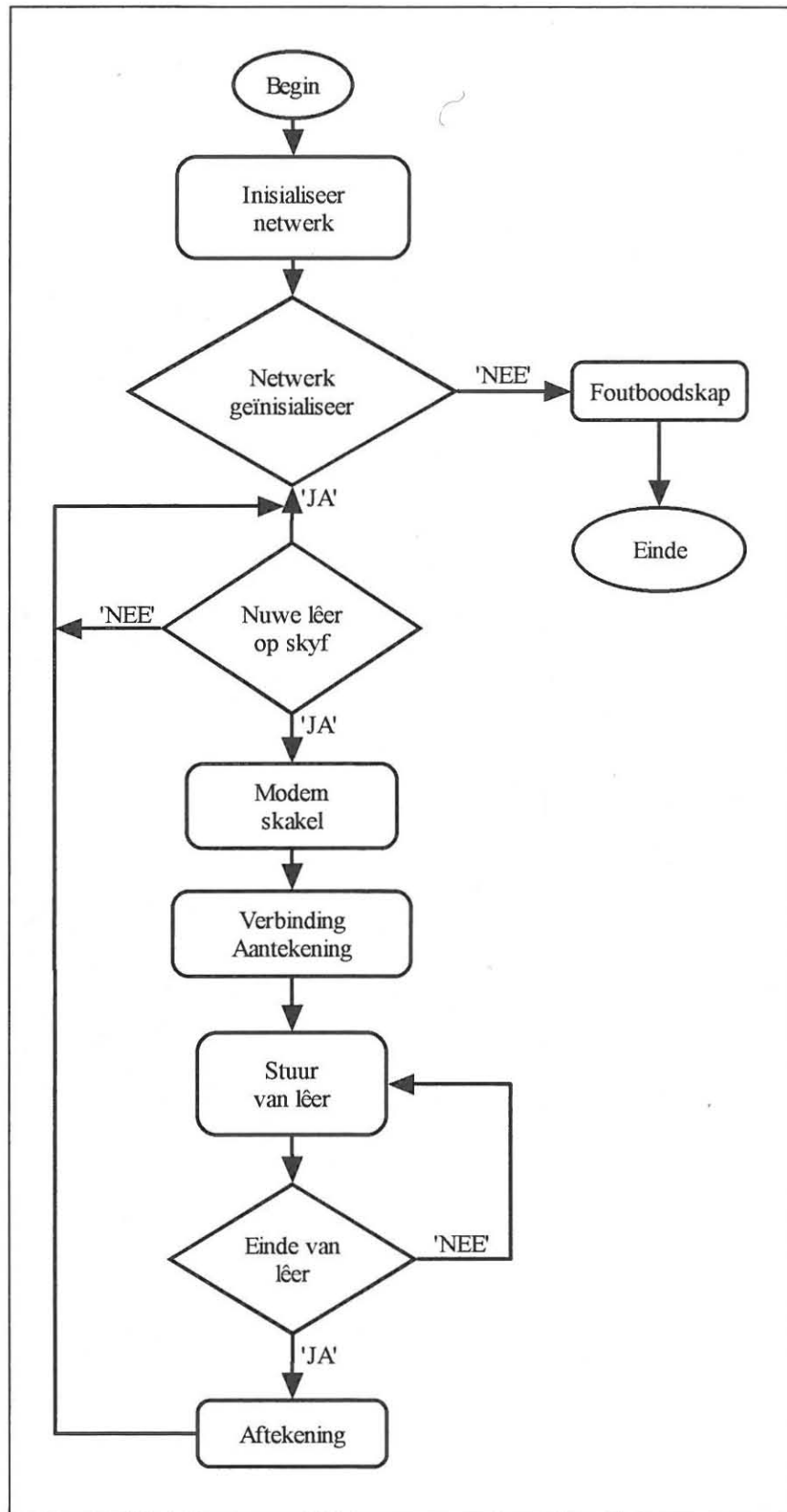
Plaas die gestoorde grafiese inligting terug nadat 'n venster toegemaak is.

◆ **void win(int x1,int y1,int x2,int y2,char colour);**

Maak 'n venster oop vir 'n teksblok op die skerm.

◆ **void block1(int x1,int y1,int x2,int y2);**

Trek 'n raamwerk om die nuwe venster wat oopgemaak is.



FIGUUR 4.2 Vloediagram van sender sagteware

4.1.3 Ontwikkeling van die modemkoppelvlak sagteware

Nadat die bogenoemde stelsel deur middel van die nulmodemkabelkoppeling deeglik getoets is, is die sagteware roetines vir die modemkoppelings geskryf. Hierdie sagteware moet die skakeling, sowel as die beëindiging van die oproep behartig. Nadat die skakeling van die nommer gedoen is en die koppeling daargestel is, moet die beheer aan die ander sagteware gegee word, sodat die sessie begin kan word deur die afstandige aantekening na die steunrekenaar. Na afloop van die sessie moet die modemsagteware weer verseker dat die koppeling verbreek word.

Die data wat gestuur word, word deur die seriale poort gestuur, wat deur 'n geïntegreerde kringbaan (GK) beheer word, wat bekend staan as 'n "Universal Asynchronous Receiver/Transmitter" (UART). Die spesifieke werking van die UART word bepaal deur die parameters wat deur die sagteware, wat besig is om op die rekenaar uitgevoer te word, beheer word. Wanneer data gestuur word, sal die UART die volgende doen:

- Ontvang 'n karakter vanaf die rekenaar.
- Skakel die karakter om in 'n reeks bisse (rangskikking).
- Stuur die seriaal gerangskikte bisse teen die seinspoed wat deur die program bepaal is (bisse per sekonde).
- Wys dat 'n volgende karakter gestuur kan word.

Wanneer die UART as 'n ontvanger gebruik word, word 'n meer komplekse taak uitgevoer:

- Ontvang seriale bisse teen die geselekteerde datatempo.
- Verifieer dat data 'n geldige struktuur is.

- Verifieer dat die pariteit, indien enige, korrek is. Indien nie, moet 'n fout aangedui word.
- Skakel data bisse om in karakter.
- Stel die karakter beskikbaar aan die rekenaar.
- Dui aan dat 'n karakter beskikbaar is.

Tabel 4.2 [13, p. 271] dui die registers op die UART aan, wat gebruik word om die modem sowel as die seriale poort te beheer.

Die sagteware hanteer die skakeling sowel as die beskikbaarmaking van die datalyn tussen die twee rekenaars. Die modem word eers geïnisialiseer waarna die skakelstring, wat die telefoonnommer bevat, na die modem gestuur word. Voordat daar instruksies na die modem gestuur word, moet die volgende twee registers in die UART gestel wees, naamlik DTR en RTS.

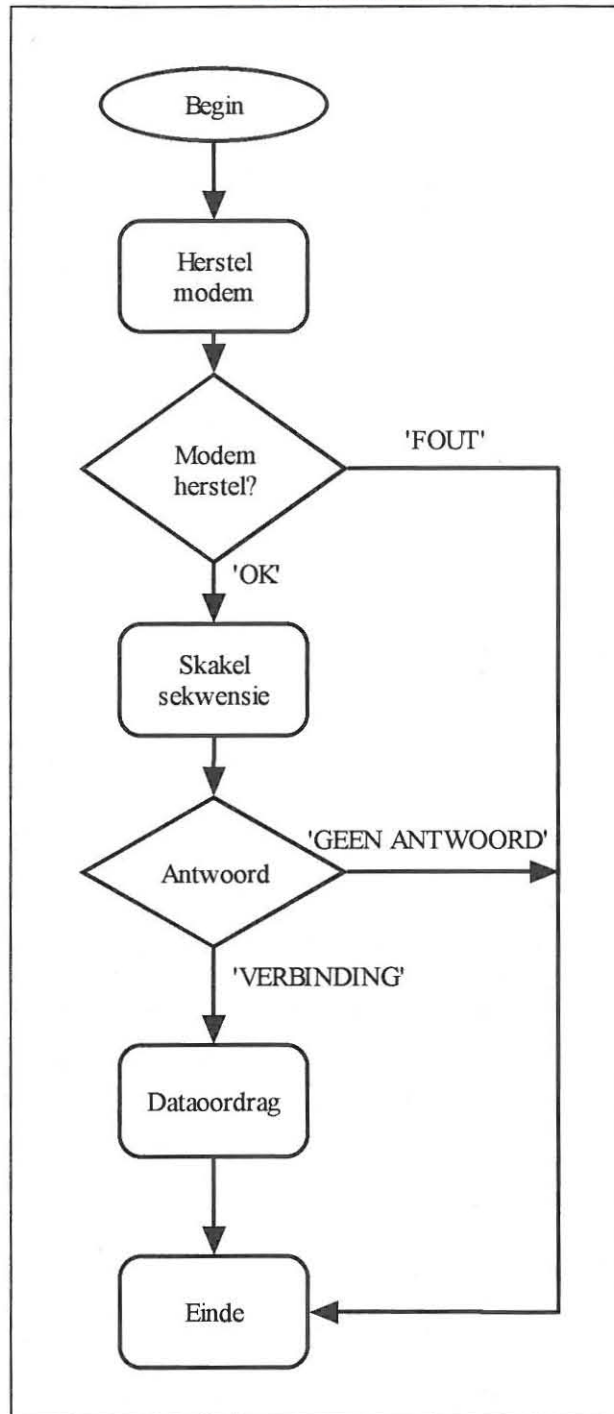
Enige instruksie wat na die modem gestuur word, word voorafgegaan deur 'AT', daarna die instruksie wat gevolg word deur 'n uitvoer (\r) instruksie. Die modemkoppelvlak sagteware voer die sekwensiële skakelproses in Figuur 4.3 stapsgewys uit:

TABEL 4.2 UART registers en adresse

Register naam	Adres/ Toegangs beberkings	Bis 7	Bis 6	Bis 5	Bis 4	Bis 3	Bis 2	Bis 1	Bis 0
Ontvanger buffer register (RBR)	Basis + 0 Leesalleen wanneer DLAB = 0	Inkomende data							
Sender houregister (THR)	Basis + 0 Skryfalleen wanneer DLAB = 0	Uitgaande data							
Onderbreking steunregister (IER)	Basis + 1 Lees/Skryf wanneer DLAB = 0					Stel data stel status onderbreking in werking (EDSSI)	Stel lyn status onderbreking in werking (ELSI)	Stel Ssend buffer leeg onderbreking in werking (ETBEI)	Stel ontvang buffer vol onderbreking in werking (ERFBI)
Onderbreking identifikasie register (IIR)	Basis + 2 Leesalleen	EIEU's in werking (16550)	EIEU's in werking (16550)				Onderbreking ID Bis 1	Onderbreking ID Bis 2	0 = Hangende onderbreking
EIEU beheer register (FCR)	Basis + 3 Lees/Skryf	Ontvanger sneller (MSB)	Ontvanger sneller (LSB)			DGT modus selekteer	Send EIEU herstel	Ontvang EIEU herstel	EIEU in werking
Lyn beheer register (LCR)	Basis + 3 Lees/Skryf	Devisor Latch toegangsbis DLAB	Send onderbreking	Stick pariteit	Ewe pariteit selekteer (EPS)	Pariteit in werking (PEN)	Stop bisse (STB)	Woord lengte selekteer Bis 1 (WLS1)	Woord lengte selekteer Bis 0 (WLS0)
Modem beheer register (MCR)	Basis + 4 Lees/Skryf				Lus	Out 2 (Stel onderbrekings in werking op PC)	Out 1 (Herstel sommige interne modems)	Request to send (RTS)	Data terminaal gereed (DTR)
Lyn status register (LSR)	Basis + 5 Lees/Skryf		Send skuiregister leeg (TSRE)	Send houregister leeg (THRE)	Breuk onderbreking (BI)	Raamfout (FE)	Pariteitsfout (PE)	Verbysteek fout (OR)	Data gereed (DR)
Modem status register (MSR)	Basis + 6 Lees/Skryf	Ontvanger lynsein opsporing (RLSD)	Lui aanwyser (RI)	Datastel gereed (DSR)	Clear to send (CTS)	Delta RLSD (DRLSD)	Daalrand lui aanwyser (TERI)	Delta DSR (DDSR)	Delta CTS (DCTS)
Kladregister (SCR)	Basis + 7 Lees/Skryf (16450/550)	Arbitrêre data							
Toestelsperring (DLL)	Basis + 0 Lees/Skryf wanneer DLAB = 1	LSB van divisor							
Toestelsperring (DLM)	Basis + 1 Lees/Skryf wanneer DLAB = 1	MSB van divisor							

TABEL 4.2 UART registers en adresse

Register naam	Adres/ Toegangs beberkings	Bis 7	Bis 6	Bis 5	Bis 4	Bis 3	Bis 2	Bis 1	Bis 0
Ontvanger buffer register (RBR)	Basis + 0 Leesalleen wanneer DLAB = 0	Inkomende data							
Sender houregister (THR)	Basis + 0 Skryfalleen wanneer DLAB = 0	Uitgaande data							
Onderbreking steunregister (IER)	Basis + 1 Lees/Skryf wanneer DLAB = 0					Stel data stel status onderbreking in werking (EDSSI)	Stel lyn status onderbreking in werking (ELSI)	Stel Ssend buffer leeg onderbreking in werking (ETBEI)	Stel ontvang buffer vol onderbreking in werking (ERFBI)
Onderbreking identifikasie register (IIR)	Basis + 2 Leesalleen	EIEU's in werking (16550)	EIEU's in werking (16550)				Onderbreking ID Bis 1	Onderbreking ID Bis 2	0 = Hangende onderbreking
EIEU beheer register (FCR)	Basis + 3 Lees/Skryf	Ontvanger sneller (MSB)	Ontvanger sneller (LSB)			DGT modus selekteer	Send EIEU herstel	Ontvang EIEU herstel	EIEU in werking
Lyn beheer register (LCR)	Basis + 3 Lees/Skryf	Devisor Latch toegangsbis DLAB	Send onderbreking	Stick pariteit	Ewe pariteit selekteer (EPS)	Pariteit in werking (PEN)	Stop bisse (STB)	Woord lengte selekteer Bis 1 (WLS1)	Woord lengte selekteer Bis 0 (WLS0)
Modem beheer register (MCR)	Basis + 4 Lees/Skryf				Lus	Out 2 (Stel onderbrekings in werking op PC)	Out 1 (Herstel sommige interne modems)	Request to send (RTS)	Data terminaal gereed (DTR)
Lyn status register (LSR)	Basis + 5 Lees/Skryf		Send skuifregister leeg (TSRE)	Send houregister leeg (THRE)	Breuk onderbreking (BI)	Raamfout (FE)	Pariteitsfout (PE)	Verbysteek fout (OR)	Data gereed (DR)
Modem status register (MSR)	Basis + 6 Lees/Skryf	Ontvanger lynsein opsporing (RLSD)	Lui aanwyser (RI)	Datastel gereed (DSR)	Clear to send (CTS)	Delta RLSD (DRLSD)	Daalrand lui aanwyser (TERI)	Delta DSR (DDSR)	Delta CTS (DCTS)
Kladregister (SCR)	Basis + 7 Lees/Skryf (16450/550)	Arbitrêre data							
Toestelsperring (DLL)	Basis + 0 Lees/Skryf wanneer DLAB = 1	LSB van divisor							
Toestelsperring (DLM)	Basis + 1 Lees/Skryf wanneer DLAB = 1	MSB van divisor							



FIGUUR 4.3 Skakelproses gedoen deur modem

Indien die fisiese verbinding suksesvol gemaak is, word die TCP/IP data netwerkkoppeling oopgemaak waarna dataoordrag kan begin. Hierdie faset handel oor die hantering van die skakelprotokol op die fisiese telefoon netwerkkoppeling wat nou die kabel vervang.

Aangesien PCMCIA modems deur die rekenaar gesien word as 'n gewone seriale poort, is geen spesiale sagteware veranderings nodig vir die gebruik van die sellulêre modems nie.

Die sagteware subroetines, wat in albei modemprogramme gebruik word is as volg (Sien Bylae 3 en 5):

◆ **void configuration(void);**

Hierdie prosedure laai die inligting vanaf die konfigurasielêer.

◆ **void Beep_Error(char *c);**

Die opwekking van 'n waarskuwingsein deur middel van die rekenaar luidspreker wanneer 'n fout met die uitvoering van die sagteware ondervind word, asook die vertoon van 'n toepaslike foutboodskap op die skerm.

◆ **void Dial_Error(char *c);**

Die opwekking van 'n waarskuwingsein deur middel van die rekenaar luidspreker wanneer 'n fout met die uitvoering van die sagteware ondervind word, asook die vertoon van 'n toepaslike foutboodskap op die skerm.

4.2 Opsomming

Hierdie hoofstuk gee 'n verduideliking van die sagteware wat ontwikkel is vir die projek.

Die volledige drukstukke van die sagteware kan in die bylae gevind word. Bylaag 2 bevat die sender sagteware, Bylaag 3 bevat die sender modemsagteware, Bylaag 4 bevat die ontvanger sagteware en Bylaag 5 bevat die ontvanger modemsagteware. Bylaag 6 omskryf die sagteware wat gebruik is om die gestuurde data te vergelyk. Bylaag 7 bevat 'n voorbeeld van die konfigurasielêer wat benodig word vir die verskaffing van die telefoonnommers en die netwerkname. Bylaag 8 bevat die globale veranderlikes vir die sender- en ontvangersagteware en Bylaag 9 bevat die globale veranderlikes vir die modemsagteware.

HOOFSTUK 5

EVALUERING VAN STELSEL

Hierdie hoofstuk bespreek die verskillende eksperimente wat gedoen is om die stelsel te toets en te evalueer. Die eksperimente is gedoen om die geskiktheid van die projek vir mediese doeleindes te bepaal, asook om die datatempo met die datatempo van kommersiële pakette te vergelyk. Daar is dus vergelykende toetse gedoen asook toetse waar die fouttempo van die stelsel bepaal is. 'n Opsomming van die resultate van die verskillende eksperimente kan in Tabel 5.4 gesien word.

5.1 Eksperiment 1: Aantekening

5.1.1 Doel: Aantekening vanaf 'n sender na die ontvanger rekenaar deur middel van 'n nulmodemkabel en die vergelyking van transmissietye tussen die ontwikkelde- en kommersieel beskikbare sagteware, soos byvoorbeeld Windows 3.1™ se terminaal sagteware.

5.1.2 Metode: Koppel twee rekenaars deur middel van 'n nulmodemkabel aan mekaar en teken aan vanaf die sender. 'n Lêer word oorgestuurd en die ontvangde data word vergelyk met die gestuurde lêer. 'n Lêer van 19845 bise is vir die doel van die eksperimente gebruik. Hierdie lêer is verkry deur 'n X-straalbeeld van die borskas van 'n persoon te versyfer en as 'n lêer te stoor. Sien Bylaag 6 vir 'n uitdruk van die sagteware wat ontwikkel is om die vergelyking tussen die oorspronklike en ontvangde lêers te doen. Die tye wat die

dataoordrag neem word aangeteken vir die ontwikkelde sagteware sowel as vir Windows 3.1™.

5.1.3 Resultate: Die data is vergelyk met die oorspronklik gestuurde data en geen foute is in die ontvangde lêers gekry nie. (Sien Tabel 5.1) Daar is tien steekproewe op bogenoemde manier gedoen waarna die gemiddelde dataoordragspoed bereken is en saamgevat word in Tabel 5.4. Uit die data soos gegee in Tabel 5.4, kan gesien word dat die terminaalsagteware se dataoordrag nie merkwaardig vinniger is as die van die ontwikkelde sagteware nie.

Aangesien hersendings lukraak voor kom op die kommunikasiemedium (nulmodemkabel), verskil die transmissietye van toets tot toets. Op die nulmodemkabel wat kort is (2 meter) is die foute baie minder as wat dit op 'n medium soos die sellulêre telefoonnetwerk sal wees.

Daar kan aanvaar word dat 'n normale verspreiding van hersendings verkry is en die gemiddelde transmissietye (μ) kan dus as volg bereken word [29, p. 49]:

$$\mu = \frac{\sum_{n=1}^x V_n}{x} \quad (5.1)$$

waar χ die aantal lesings en V_n die lesings is.

Hieruit kan die standaardafwyking (σ) as volg bepaal word [29, p. 51]:

$$\sigma = \frac{\sum_n^x (V_n - \mu)}{x} \quad (5.2)$$

Variansie (σ^2) kan dus nou uit bogenoemde bereken word [29, p. 51]:

$$\sigma^2 = \frac{\sum_n^x (Vn - \mu)^2}{x} \quad (5.3)$$

TABEL 5.1 Gemiddelde tye, Standaard afwyking en variansie van eksperiment 1

Medium	Gemiddelde tye (μ)	Standaard- afwyking (σ)	Variansie (σ^2)
TCP/IP	33 sekondes	$1,4 \times 10^{-15}$	1,49
Terminaal	28 sekondes	$-1,14 \times 10^{-15}$	3,44

5.1.4 Gevolgtrekking: Die dataoordrag is stadiger as die wat gedoen is deur middel van Windows 3.1™ se terminaal sagteware, aangesien die dataoordrag met behulp van die ontwikkelde sagteware oor 'n TCP/IP netwerkkoppeling geskied. Die pakkies data wat gestuur word, bevat inligting soos byvoorbeeld die TCP en IP kopetikette en ook ander data soos getoon in Tabel 4.1. Die transmissietyd vir elke nuwe oordrag is uniek aangesien foute ter enigertyd op die medium kan voor kom en 'n blok data dus weer gestuur sal moet word om te verseker dat die ontvangde data foutloos sal wees. Sien Tabel 5.1 vir 'n uiteensetting van die gemiddelde waardes, die standaardafwyking en variansies ten opsigte van transmissietye. Verwys ook na Tabel 5.4 vir 'n samevatting van die oordragtye en datagroottes.

5.2 Eksperiment 2: Dataoordrag en modemsakelings

5.2.1 Doel: Die doel van hierdie eksperiment is die toetsing van die dataoordrag betroubaarheid, oordragtempo's en die modemsakelsagteware deur gebruik te maak van gewone modems wat gekoppel is aan die publieke skakeltelefoonnetwerk (PSTN).

5.2.2 Metode: Twee rekenaars word aan gewone modems gekoppel. 'n Creative en Microcom modem is onderskeidelik gebruik vir hierdie eksperiment. Die sagteware wat ontwikkel is, word in 'n lus gesit sodat tien lêers gestuur sal word, nadat die skakeling deur die modem gedoen is en 'n netwerkkoppeling bewerkstellig is. Nadat 'n lêer gestuur is, word dit met die oorspronklike lêer vergelyk om sodoende fouttempo's te kan bepaal. Die sagteware wat vir hierdie doel ontwikkel is, kan in Bylaag 6 gesien word.

5.2.3 Resultate: Die foutpersentasie in die gestuurde lêers is 0%. Die rede hiervoor is dat alle foute deur die netwerksagteware gekorrigeer word. Hierdie korreksie van die data word gedoen deur die kontrolesomme van die TCP en IP kopetikette van elke pakkie te vergelyk met die nuwe ontvangde data. Indien foutiewe data ontvang is, word die pakkie data weer gestuur totdat die data reg ontvang is.

TABEL 5.2 Gemiddelde tye, Standaard afwyking en variansie van eksperiment 2

Medium	Gemiddelde tye (μ)	Standaard-afwyking (σ)	Variansie (σ^2)
TCP/IP	107 sekondes	$5,7 \times 10^{-15}$	6,84
Terminaal	111 sekondes	$2,8 \times 10^{-15}$	1,61

5.2.4 Gevolgtrekking: Alle data en tye is vergelyk (Sien tabel 5.4) en data is foutloos oorgedra na die ontvanger rekenaar. Die modemsakelsagteware het die skakeling sowel as die verbinding tussen die twee rekenaars gemaak, waarna die netwerksessie begin is. Daar is aangeteken, die lêers is gestuur en die sagteware het daarna weer die verbinding beëindig.

Hoewel die transmissietyd van die ontwikkelde sagteware vinniger is as die transmissietyd van Windows 3.1™, kan dit waarskynlik aan die feit toegeskryf word dat die werklike transmissietyd van Windows 3.1™ nie so maklik bepaal kan word soos met die ontwikkelde sagteware nie, aangesien die ontwikkelde sagteware die transmissietyd vertoon nadat die lêer gestuur is. Die transmissietyd van die ontwikkelde sagteware kan dus baie akkuraat bepaal word, terwyl daar nie presies bepaal kan word hoe lank die werklike dataoordrag deur die terminaal sagteware is nie.

5.3 Eksperiment 3: Sellulêre koppeling

5.3.1 Doel: Die koppeling van 'n sellulêre modem en die toetsing van die skakelsagteware en ook die bepaling van fouttempo's op die bogenoemde metode.

5.3.2 Metode: Een rekenaar word aan 'n gewone PSTN modem gekoppel terwyl 'n draagbare rekenaar aan 'n PCMCIA sellulêre modem gekoppel word. 'n Siemens S3 sellulêre telefoon en modem is gebruik saam met die draagbare rekenaar, terwyl 'n Microcom modem aan die ander rekenaar gekoppel is. Aangesien die vinnigste dataspoed van die sellulêre modem 9600 baud is, is alle dataoordragtempo's teen hierdie baudspoed gedoen. Om die vergelyking tussen die mediums te vergemaklik is alle toetse teen 'n baudspoed van 9600 gedoen.

5.3.3 Resultate: Die resultate van die eksperiment kan in Tabel 5.3 gesien word.

Indien daar dus nou logies hierna gekyk word, kan gesien word dat pakkies data, wat gestuur word met behulp van die ontwikkelde netwerksagteware, ongeveer elke 6 sekondes op die hardeskyf gestoor word. Daar kan gesien word dat indien 'n lêer met 'n grootte van 19845 greep gestuur word en dit 261 sekondes neem om te stuur, dit min of meer 44 skryfsiklusse sal neem om die volle lêer te stuur. Sien onderstaande formule:

$$Siklusse = \frac{261sek}{6sek} = 43,5 \quad (5.4)$$

Indien die data dus nou opgedeel word in 44 pakkies wat gestuur moet word, kan gesien word dat die waarde wat verkry word min of meer ooreenstem met die pakkies se grootte (499 greep) soos wat deur die sagteware bepaal is. Sien onderstaande vergelyking:

$$Grootte = \frac{19845}{44} = 451 \quad (5.5)$$

Aangesien 6 sekondes slegs 'n gemiddelde waarde is, verskil die werklike pakkiegroottes met die grootte van die berekenings.

TABEL 5.3 Gemiddelde tye, Standaard afwyking en variansie van eksperiment 3

Medium	Gemiddelde tye (μ)	Standaard- afwyking (σ)	Variansie (σ^2)
TCP/IP	261 sekondes	$-1,1 \times 10^{-14}$	46,36
Terminaal	223 sekondes	$-1,1 \times 10^{-14}$	24,76

5.3.4 Gevolgtrekking: Die verskil in transmissietye kan waarskynlik toegeskryf word aan die feit dat wanneer op 'n medium soos die sellulêre telefoonnetwerk gewerk word, waar baie steurings en tydgleufonderbrekinge op die kommunikasie kanaal voor kom, moet pakkies data meer gereeld, as gevolg van foute hersend word. Aangesien die pakkies data relatief groot (499 greep) is, word baie tyd spandeer om foutief ontvangde data weer te hersend. 'n Balans tussen die pakkies se grootte moet egter gehandhaaf word, aangesien elke pakkie van 'n TCP sowel as 'n IP kopetiket voorsien word. Indien die pakkies se

grootte dus te klein gemaak word, word baie tyd gemors met die versending van die TCP en IP kopetikette, wat voor elke pakkie gevoeg word.

TABEL 5.4 Eksperiment resultate

Eksperiment	Grootte van lêer	Gemiddelde transmissietye		Foute in ontvangde lêer	
		TCP/IP	Terminaal	TCP/IP	Terminaal
Eksperiment 1 Nulmodem	19 845 greep	33 sekondes (Baud: 9600)	28 sekondes (Baud: 9600)	0	0
Eksperiment 2 Gewone modems	19 845 greep	107 sekondes (Baud: 9600)	111 sekondes (Baud: 9600)	0	0
Eksperiment 3 Sellulêre modem	19845 greep	261 sekondes (Baud: 9600)	223 sekondes (Baud: 9600)	0	0

5.4 Gevolgtrekking

Soos reeds genoem is die dataoordrag stadiger as met die beproefde kommersiële sagteware, maar daar moet onthou word dat die sagteware wat ontwikkel is, gebruik maak van 'n netwerkkoppeling en dat die sagteware ontwikkel is om die oordrag van lêers eenvoudig te maak.

Vir die doel van hierdie projek word die resultate wat met die nulmodemkabel verkry is, geïgnoreer aangesien die medium nie geskik is vir die afstandige oordrag van data nie. Uit Tabel 5.4 kan gesien word dat die oordragtyd met behulp van die sellulêre koppeling stadiger is as die oordrag op gewone telefoon koppelings. Hierdie verskil in oordragspoed kan toegeskryf word aan die feit dat meer steurings op 'n sellulêre koppeling voorkom as op 'n gewone telefoon koppeling. Hoewel die data stadiger oorgedra word met behulp van die sellulêre koppeling, moet onthou word dat die data wel foutloos ontvang word.

Die grootste nadele van die sellulêre stelsel is dus:

- Die koste verbonde aan die oproep. 'n Langer oproep word gemaak as in die geval van die gewone modems. Uit die resultate in Tabel 5.4 kan gesien word dat dit 2,44 keer langer neem om die data op 'n sellulêrekoppeling oor te dra.
- Die tyd wat die ontvanger rekenaar besig gehou word. Hoe langer hierdie rekenaar besig is met die oordrag van 'n lêer, hoe minder gebruikers kan gebruik maak van die stelsel op een dag. 'n Sellulêre koppeling sal dus die stelsel langer beset as wat 'n gewone modem koppeling sal (Sien Tabel 5.4).

Voordele van die sellulêre stelsel is:

- Geen fisiese verbinding word benodig vir die oordrag van die data nie.
- Dekking sal verbeter.
- Die TCP/IP verseker getroue kommunikasie al kan die kanaal kwaliteit verlaag.

Indien die stelsel dus vanuit 'n mobiele kliniek gebruik sal word, sal die sellulêre verbinding gebruik word, maar indien die fasiliteite beskikbaar is, moet gebruik gemaak word van 'n gewone PSTN modem. Hierdie metode sal dus geld aan oproepkoste spaar sowel as netwerktyd, ten opsigte van die tyd wat die ontvanger spandeer om een lêer oor te dra.

Daar moet egter daarop gelet word dat die hoofdoel van die studie is om 'n betroubare netwerkkoppeling aan mobiele mediese eenhede te verskaf. Aangesien 'n lae fouttempo verkry is, en TCP/IP netwerkkoppeling die foutkorreksie en hertransmissie van blokke data waarin foute voorgekom het,

behartig, is die stelsel geskik vir die oordrag van mediese data vanaf 'n mobiele mediese eenheid.

HOOFSTUK 6

SAMEVATTING

Die hoofdoel van hierdie studie was die daarstelling van 'n betroubare netwerkkoppeling tussen 'n mobiele mediese eenheid en 'n sentrale mediese sentrum, soos byvoorbeeld 'n groot hospitaal. Hierdie netwerkkoppeling moet 'n betroubare medium vir die oordrag van mediese data verskaf. Hierdie tipe van dataoordrag word al hoe belangriker, teneinde die doeltreffende voorsiening van mediese dienste, aan veral agtergeblewe landelike gemeenskappe, te verseker.

Hoofstuk 2 behandel die verskillende kommunikasiemediums wat beskikbaar is vir die voorsiening van 'n mobiele mediese eenheid. Hierin word die werking, sowel as die voor- en nadele van sellulêre tegnologie, satelliet- en radio toerusting bespreek.

Die beskikbare netwerktegnologie, sowel as protokolle word in hoofstuk 3 behandel. Alhoewel alle netwerktipes en protokolle nie behandel kan word nie, word 'n redelik breedvoerige oorsig, sowel as die redes vir die keuse van TCP/IP as protokol, gegee.

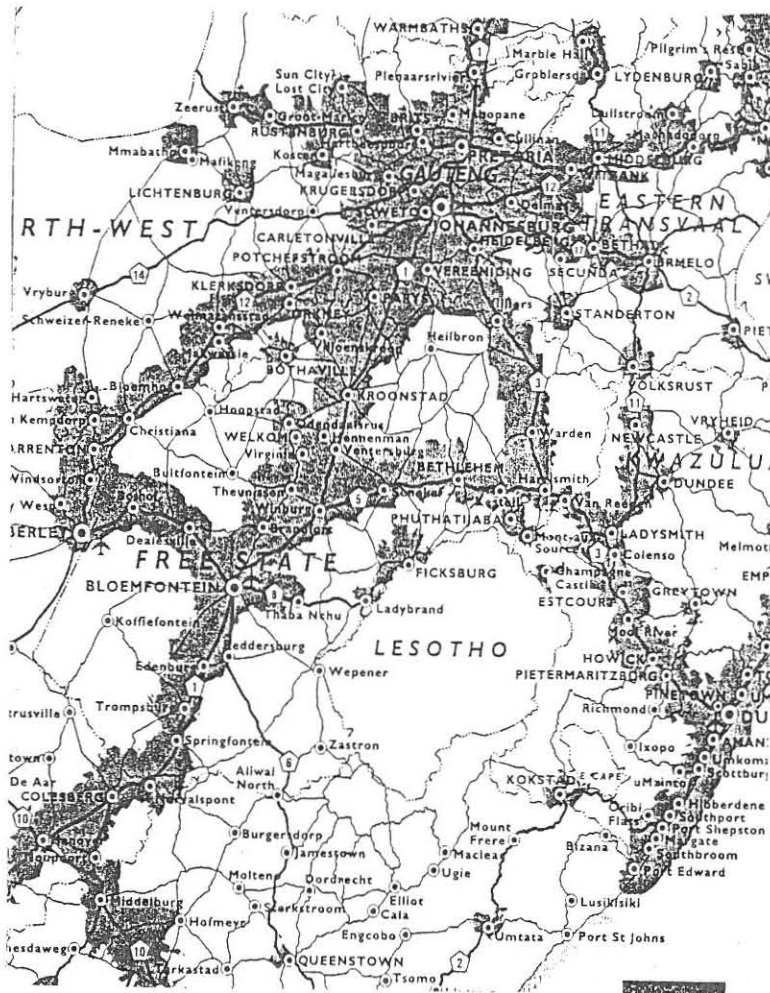
Sagteware ontwikkeling word behandel in hoofstuk 4. Alle sagteware is in C++ geskryf en Borland Turbo C++ is gebruik om die programme te kompilleer.

Hoofstuk 5 handel oor die verskillende eksperimente wat gedoen is om die stelsel te toets en onderskeie fouttempo's te bepaal.

Verdere navorsingsvoorstel

Verdere navorsing kan gedoen word om die transmissietye te verkort. Dit kan gedoen word deur 'n beter kanaal daar te stel vir die stuur van die data, sodat minder foute tydens die transmissie voor kom en ook beter foutkorreksie, sodat blokke data nie hersend hoef te word nie.

BYLAAG 1



MTN Dekkingskaart vir 1995 van die Vrystaat
(Die donker gedeeltes dui aan waar dekking beskikbaar is.)

BYLAAG 2 - HOST.C

```
/**
//
//          FILE: HOST.C
//          TCP/IP HOST
//          AUTHOR: PS VELDTSMAN
//
//          IMPORTANT
//          Use SMALL model to compile
/**

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <string.h>
#include <time.h>
#include <conio.h>
#include "api.h"

#define BUFSIZE 2000

/**
//          GLOBAL DECLERATIONS FOR WINDOWS
/**

#define left_top      201
#define left_bottom  200
#define right_top    187
#define right_bottom 188
#define horizontal   205
#define vertical     186

char window_buffer[13][3000],colour;          /* global declarations */
int window_number=0;

void win(int x1,int y1,int x2,int y2,char colour);          /* declare sub-routines */
void block1(int x1,int y1,int x2,int y2);
void open_window(char x11,char y11,char x22,char y22);
void close_window(char x11,char y11,char x22,char y22);

/**
//          GLOBAL DECLERATIONS
/**

#define NIL    0

#define MaxConns 2

typedef char tStr80[79];
typedef char str499[499];

enum flag1    { offline,
               LAN,
               disconnect,
               reqlist,
               rxlock,
               idle,
```

```

connectd,
reqdlist,
rxlist,
rxdlist,
reqlist1,
reqlist2,
reqlist3
};

typedef enum flag1 teStatus;

typedef struct { char Hostname[79];
                teStatus Status;
                char Msg[499];
                } trNet;

trNet Network;

tStr80 MyNetname;
tStr80 UserName;

void NetInit(void);
void User(void);
void Menu(void);
void WarningSound(void);
void TimeSlice(void);
int open_tcp(TCP_OP_P op);
void API_Loaded(void);
void close_tcp(unsigned tcbo);
void del_tcp(unsigned tcbo);
char *err(unsigned errcode);

/*****
//          GLOBAL DECLERATIONS TCPBIOS
*****/

enum TEXFER {elCompl, elBusy, elError, elClosed};
typedef enum TEXFER teXfer;

typedef struct TrXfer { teXfer eState;
                       char *pBuf;
                       unsigned cBuf;
                       cLen;
                       cNow;
                       cRdy;
                       cErr;
                       } trXfer;

char kzLenID[14] = "BytesToExpect:";

typedef struct trLen { char zID[15];
                     unsigned cLen;
                     unsigned J_Value;
                     char File_Name[13];
                     unsigned Command;
                     } trLen;

char *File_Name_Ptr;

```



```

unsigned long int GetIPAddr(char szAddr[]);
unsigned int TCPOpen(TCP_OP_P op, unsigned far *psegTCB, unsigned far *pofsTCB);
void APIAlive(void);
void TxLenData (unsigned segTCP, unsigned ofsTCP, struct TrXfer *ptrNCB);
void RxLenData (unsigned ofsTCP, struct TrXfer *ptrNCB);
void SetLenData (struct TrXfer *ptrNCB, char *ptradbBuf, unsigned cSize);
unsigned TCPRd(unsigned ofsTCB, struct trLen *pachBuf, unsigned *pcBuf);
unsigned TCPRd1(unsigned ofsTCB, char *pachBuf, unsigned *pcBuf);
unsigned TCPWr(unsigned ofsTCB, struct trLen far *pachBuf, unsigned *pcBuf);
unsigned TCPWr1(unsigned ofsTCB, char *pachBuf, unsigned *pcBuf);

```

```

//*****
//      GLOBAL DECLERATIONS REMTCPB
//*****

```

```

const Busy = 0;
const Ok   = 1;
const Err  = 2;

```

```
trLen rLen;
```

```

enum flag {TRUE,FALSE};
enum flag bInit;

```

```

unsigned int segTCP,ofsTCP;
unsigned RxLen;

```

```

typedef struct {  int ofsConn;
                  } arTCB[32];

```

```

struct TCPRec {  unsigned ofsConn;
                 trXfer rTxNCB;
                 trXfer rRxNCB;
                 }          arTCP[MaxConns];

```

```

unsigned int NetConnect(char Host[], unsigned Index);
unsigned int NetConnected(unsigned int Index);
unsigned TxBuffer(unsigned Index, char *ptrTxBuf, unsigned TxLen);
unsigned TxdBuffer(unsigned Index);
unsigned RxBuffer(unsigned Index, char *ptrRxBuf);
unsigned RxdBuffer(unsigned Index, unsigned *ptrRxLen);

```

```

//*****
//      GLOBAL DECLERATIONS
//*****

```

```
unsigned segTCB,ofsTCB;
```

```
FILE *fp_in;
```

```

unsigned serv_tcb;
unsigned numcon;           // number of connections
unsigned tcbs;
unsigned long LastActive;

```

```

char buf[BUFSIZE];
char err_str[100];

```

char error;

TCP_OP_P op = {0,0x4446,0,TCP_ACTIVE,0,NIL,NIL,NIL,0,0};

```

//*****
//      PROCEDURES TCPBIOS
//*****

```

unsigned TCPRd(ofsTCB, pachBuf, pcBuf)

```

unsigned ofsTCB;
struct trLen *pachBuf;
unsigned *pcBuf;
{
    unsigned temp1,temp2;
    _SI = FP_OFF(pachBuf);
    _DX = FP_SEG(pachBuf);
    _CX = *pcBuf;
    _BX = ofsTCB;
    _AH = API_TCPREAD;
    geninterrupt(API_INT);
    temp1 = _CX;
    temp2 = _AX;
    *pcBuf = temp1;
    return(temp2);
}

```

unsigned TCPRd1(ofsTCB, pachBuf, pcBuf)

```

unsigned ofsTCB;
char *pachBuf;
unsigned *pcBuf;
{
    unsigned temp1,temp2;
    _SI = FP_OFF(pachBuf);
    _DX = FP_SEG(pachBuf);
    _CX = *pcBuf;
    _BX = ofsTCB;
    _AH = API_TCPREAD;
    geninterrupt(API_INT);
    temp1 = _CX;
    temp2 = _AX;
    *pcBuf = temp1;
    return(temp2);
}

```

unsigned TCPWr(ofsTCB, pachBuf, pcBuf)

```

unsigned ofsTCB;
struct trLen far *pachBuf;
unsigned *pcBuf;
{
    unsigned temp1, temp2;
    _SI = FP_OFF(pachBuf);
    _DX = FP_SEG(pachBuf);
    _CX = *pcBuf;
    _BX = ofsTCB;
    _AH = API_TCPWRITE;
    _AL = 0;
    geninterrupt(API_INT);
    temp1 = _CX;
}

```

```
temp2 = _AX;
*pcBuf = temp1;
return (temp2);
}
```

```
unsigned TCPWr1(ofsTCB, pachBuf, pcBuf)
unsigned ofsTCB;
char *pachBuf;
unsigned *pcBuf;
{
    unsigned temp1, temp2;
    _SI = FP_OFF(pachBuf);
    _DX = FP_SEG(pachBuf);
    _CX = *pcBuf;
    _BX = ofsTCB;
    _AH = API_TCPWRITE;
    _AL = 0;
    geninterrupt(API_INT);
    temp1 = _CX;
    temp2 = _AX;
    *pcBuf = temp1;
    return (temp2);
}
```

```
unsigned int TCPOpen(TCP_OP_P op, unsigned far *psegTCB, unsigned far *pofsTCB)
{
    unsigned temp1, temp2, temp3;
    _CX = FP_OFF(&op);
    _BX = FP_SEG(&op);
    _AH = API_TCPOPEN;
    geninterrupt(API_INT);

    temp1 = _BX;
    temp2 = _CX;
    temp3 = _AX;
    *psegTCB = temp1;
    *pofsTCB = temp2;
    return (temp3);
}
```

```
unsigned long int GetIPAddr(char szAddr[])
{
    long cx,dx;
    _BX= FP_SEG(szAddr);
    _CX= FP_OFF(szAddr);
    _AH= API_GETIPADR;
    geninterrupt(API_INT);
    cx=_CX;
    dx=_DX;
    cx=(cx<<16);
    cx=cx+dx;
    return(cx);
}
```

```
void APIAlive(void)
{
    _AH = API_NOOP;
    geninterrupt(API_INT);
}
```

```

}

void TxLenData(segTCP, ofsTCP, ptrNCB)
unsigned segTCP;
unsigned ofsTCP;
struct TrXfer *ptrNCB;
{
    TCB far *TCBPtr;
    static unsigned cSnd, cTxd;
    if (ptrNCB -> eState == eIBusy)
    {
        if (ptrNCB -> cLen == 0)                                /* Len not yet sent */
        {
            rLen.cLen = ptrNCB -> cBuf;
            strcpy (rLen.zID, kzLenID);
            cTxd = sizeof(rLen);

            ptrNCB -> cErr = TCPWr(ofsTCP, &rLen, &cTxd);

            switch (ptrNCB -> cErr)
            { case NONE: { if (cTxd == sizeof(rLen))
                { ptrNCB -> cNow = 0;
                  ptrNCB -> cLen = ptrNCB -> cBuf;
                  if (ptrNCB -> cBuf == 0)
                      ptrNCB -> eState = eICompl;
                }
                TCBPtr = MK_FP(segTCP,ofsTCP);
                if (TCBPtr -> state == CLOSED)
                    ptrNCB -> eState = eIClosed;
                break;
            }

            case NO_CONN : { ptrNCB -> eState = eIClosed;
                            break;
                        }

            case CON_CLOS : { ptrNCB -> eState = eIClosed;
                              break;
                          }

            default:      { ptrNCB -> eState = eIError;
                          break;
                        }

            }
            if (ptrNCB -> eState != eIBusy)
                return;
        }
    }

    if (ptrNCB -> cLen > 0)                                    /* Len sent */
    {
        cSnd = ptrNCB -> cBuf - ptrNCB -> cNow;
        TCBPtr = MK_FP(segTCP,ofsTCP);
        if (cSnd > TCBPtr -> mss)
            cSnd = TCBPtr -> mss;
        if (cSnd <= ptrNCB -> cRdy)
        {
            cTxd = cSnd;
        }
    }
}

```



```
ptrNCB -> cErr = TCPWr1(ofsTCP, ptrNCB -> pBuffer = &Network.Msg[ptrNCB -> cNow],
&cTxd);
switch (ptrNCB -> cErr)
{
case NONE: { ptrNCB -> cNow = ptrNCB -> cNow + cTxd;
             if (ptrNCB -> cNow >= ptrNCB -> cLen)
               ptrNCB -> eState = eICompl;
             TCBPtr = MK_FP(segTCP, ofsTCP);
             if (TCBPtr -> state == CLOSED)
               ptrNCB -> eState = eIClosed;
             break;
           }

case NO_SPACE: { ptrNCB -> cNow = ptrNCB -> cNow + cTxd;
                 if (ptrNCB -> cNow >= ptrNCB -> cLen)
                   ptrNCB -> eState = eICompl;
                 TCBPtr = MK_FP(segTCP, ofsTCP);
                 if (TCBPtr -> state == CLOSED)
                   ptrNCB -> eState = eIClosed;
                 break;
               }

case NO_CONN: { ptrNCB -> eState = eIClosed;
                break;
              }

case CON_CLOS: { ptrNCB -> eState = eIClosed;
                 break;
               }

default: ptrNCB -> eState = eIError;
        }
}
}
```

```
void SetLenData (ptrNCB, ptradbBuf, cSize)
struct TrXfer *ptrNCB;
char *ptradbBuf;
unsigned cSize;
{
ptrNCB -> eState = eIBusy;
ptrNCB -> pBuffer = ptradbBuf;                /*tadbB[0];*/
ptrNCB -> cBuf = cSize;
ptrNCB -> cLen = 0;
ptrNCB -> cNow = 0;
ptrNCB -> cErr = 0;
ptrNCB -> cRdy = 65535;
}
```

```
void RxLenData(ofsTCP, ptrNCB)
unsigned ofsTCP;
struct TrXfer *ptrNCB;
{
static unsigned cRxd;
```

```

if (ptrNCB -> eState == elBusy)
{
    if (ptrNCB -> cLen == 0)                                     /* Len not yet found */
    {
        cRxd = sizeof(trLen);
        ptrNCB -> cErr = TCPRd(ofsTCP, &rLen, &cRxd);
        switch (ptrNCB -> cErr)
        {
            case NONE: { if (cRxd == sizeof(trLen))
                if (strcmp(rLen.zID, kzLenID) == 0)
                { ptrNCB -> cLen = rLen.cLen;
                    ptrNCB -> cNow = 0;
                    if (ptrNCB -> cLen == 0)
                        ptrNCB -> eState = elCompl;
                    break;
                }
                break;
            }
            case WOULDBLK: { break;
            }
            case NO_SPACE: { break;
            }
            case NO_CONN : { ptrNCB -> eState = elClosed;
                break;
            }
            case CON_CLOS: { ptrNCB -> eState = elClosed;
                break;
            }
            default : ptrNCB -> eState = elError;
                break;
        }
    }

    if (ptrNCB -> eState != elBusy)
        return;
}

if (ptrNCB -> cLen > 0)                                       /* Len found */
{
    if (ptrNCB -> cRdy > 0)
    {
        cRxd = ptrNCB -> cLen - ptrNCB -> cNow;
        ptrNCB -> cErr = TCPRd1(ofsTCP, ptrNCB -> pBuf = &Network.Msg[ptrNCB -> cNow],
&cRxd);

        switch (ptrNCB -> cErr)
        {
            case NONE: { ptrNCB -> cNow = ptrNCB -> cNow + cRxd;
                if (cRxd == 0)                                     /* Remote closed session */
                    ptrNCB -> eState = elClosed;
                else
                    if (ptrNCB -> cNow >= ptrNCB -> cLen)

```

```

        ptrNCB -> eState = elCompl;
        break;
    }

    case WOULDBLK: break;

    case NO_SPACE: break;

    case NO_CONN : { ptrNCB -> eState = elClosed;
                    break;
                  }

    case CON_CLOS: { ptrNCB -> eState = elClosed;
                    break;
                  }

    default      : ptrNCB -> eState = elError;
    }
}
}
}
}

/*****
//      PROCEDURES REMTCPB
*****/

unsigned int NetConnect(char Host[], unsigned Index)
{
    unsigned long int lcRemAddr;
    long int cErr;
    if (bInit)
    {
        lcRemAddr=GetIPAddr(Host);
        if (lcRemAddr == 0)
            return(Err);
        op.rem_ip_ad = lcRemAddr;
        cErr=TCPOpen(op, &segTCP, &arTCP[Index].ofsConn);

        if (cErr>0)
            return Err;
        else
            return(Ok);
    }
    return (Ok);
}

unsigned int NetConnected(unsigned int Index)
{
    TCB far *tcb = MK_FP(segTCP, arTCP[Index].ofsConn);
    APIAlive();
    switch (tcb->state)
    {
        case ESTABLISHED: return(Ok);
        case LISTEN:      return(Busy);
        case SYN_SENT:    return(Busy);
        case SYN_RECEIVED: return(Busy);
    }
}

```

```

        default:                return(Busy);
    }
}

unsigned TxBuffer(Index, ptrTxBuf, TxLen)
unsigned Index;
char *ptrTxBuf;
unsigned TxLen;

{
    SetLenData(&arTCP[Index].rTxNCB, ptrTxBuf, TxLen);
    return(Ok);
}

unsigned TxdBuffer(unsigned Index)
{
    TxLenData(segTCP, arTCP[Index].ofsConn, &arTCP[Index].rTxNCB);
    switch (arTCP[Index].rTxNCB.eState)
    {
        case elBusy : break;
        case elCompl: return(Ok);
        default:     return(Err);
    }
    return(Busy);
}

unsigned RxBuffer(Index, ptrRxBuf)
unsigned Index;
char *ptrRxBuf;
{
    SetLenData(&arTCP[Index].rRxNCB, ptrRxBuf, sizeof(Network.Msg));
    return(Ok);
}

unsigned RxdBuffer(Index, ptrRxLen)
unsigned Index;
unsigned *ptrRxLen;
{
    RxLenData(arTCP[Index].ofsConn, &arTCP[Index].rRxNCB);
    *ptrRxLen = arTCP[Index].rRxNCB.cNow;
    switch (arTCP[Index].rRxNCB.eState)
    {
        case elBusy : break;
        case elCompl: return(Ok);
        default    : { /* Closed, Err */
                        PostTcp();
                        return(Err);
                    }
    }
    return(Busy);
}

/*****
//      PROCEDURE
*****/

int open_tcp(TCP_OP_P op)
{

```



```

unsigned errcode;
_CX = FP_OFF(&op);
_BX = FP_SEG(&op);
_AH = API_TCPOPEN;
geninterrupt(API_INT);
errcode = _AX;
ofsTCP = _CX;
segTCP = _BX;
if (ofsTCP == 0)
{
    printf("Error on TCP_OPEN: %s\n",err(errcode));
    exit(1);
}
return (errcode);
}

void TimeSlice(void)
{
    unsigned int Len,i;
    enum flag2 {Busy,Ok,Err};
    enum flag2 ret;

    static unsigned temp300 = 0,
                    temp400 = 0;
    static float    temp200 = 0,
                    temp201 = 0;
    static int      *temp100;

    static time_t tstart, tstop;

    int blocks;

    i=1;

    switch (Network.Status)
    {
        case offline: break;
        case LAN:{ ret=NetConnect(Network.Hostname,i);
                    switch (ret)
                    {
                        case Busy: break;
                        case Ok :{ open_window (30,10,56,13);
                                    win (30,10,56,13,CYAN);
                                    gotoxy(4,2);
                                    printf("Connecting.....");
                                    Network.Status=connectd;
                                    break;
                                }
                        }
                    case Err :{ gotoxy(6,3);
                                printf("Connect Error!");
                                Network.Status=offline;
                                delay(2000);
                                close_window (30,10,56,13);
                                window (1,5,80,22);
                                break;
                            }
                    }
        default: break;
    }
}

```

```

break;
}
case connectd:{ ret=NetConnected(i);
switch (ret)
{
case Busy: break;
case Ok :{ gotoxy(8,3);
printf("Connected");
Network.Status=rxlist;
delay(1000);
close_window (30,10,56,13);
window (1,5,80,22);
break;
}
case Err :{ gotoxy(6,3);
printf("Connect Error!");
Network.Status=offline;
delay(2000);
close_window (30,10,56,13);
window (1,5,80,22);
break;
}
default: Network.Status = LAN;
break;
}
break;
}

case reqlist1:{ strcpy (Network.Msg,"");
tstart = time(NULL);
if ((fp_in = fopen(rLen.File_Name, "rb")) == NULL)
{ open_window (20,8,60,11);
_setcursortype (_NOCURSOR);
win (20,8,60,11,RED);
textcolor (WHITE);
gotoxy (18,1);
cprintf (" FOUT ");
textcolor (BLACK);
gotoxy(7,2);
printf ("Error opening source file");
gotoxy (15,3);
printf ("Press a key");
Network.Status = idle;
getch();
close_window (20,8,60,11);
close_window (27,10,60,13);
window (1,5,80,22);
break;
}
rLen.Command = 0;
// Determine size of file
for (; ;)
{ *temp100 = fgetc(fp_in);
if (*temp100 == EOF)
{
temp201 = temp200;
Network.Status = reqlist2;
rewind (fp_in);
}
}
}

```

```

        break;
    }
    else
    {
        temp200++;
    }
}
break;
}

case reqlist2:{ strcpy(Network.Msg,"");
    rLen.J_Value = 499;
    for (temp300=0; temp300 <= 498; temp300++)
    {
        if (fread(Network.Msg+temp300, 1, 1, fp_in) != 1)
        {
            // printf("Read Error\n");
            fclose (fp_in);
            rLen.Command = 2;
            rLen.J_Value = temp300;
            temp300 = 499;
            Network.Status = reqlist3;
            break;
        }
    }
    Network.Status = reqlist3;
    break;
}

case reqlist3:{ ret = TxBuffer(i, Network.Msg, sizeof(Network.Msg));
    switch (ret)
    {
        case Busy : break;

        case Ok  :{ Network.Status = reqdlist;
                    break;
                }
        case Err :{ Network.Status = disconnect;
                    break;
                }
        default  : break;
    }
    break;
}

case reqlist:{ ret = TxBuffer(i, Network.Msg, sizeof(Network.Msg));
    switch (ret)
    {
        case Busy : break;
        case Ok  :{ Network.Status = reqdlist;
                    break;
                }
        case Err :{ Network.Status = disconnect;
                    break;
                }
        default  : break;
    }
    break;
}

```

```

    }
case reqdlist: { ret = TxdBuffer(i);
    switch (ret)
    {
        case Busy : break;
        case Ok  : { Network.Status = rxlist;
                    break;
                }
        case Err  : { Network.Status=disconnect;
                    break;
                }
        default  : break;
    }
    break;
}

case rxlist : { strcpy (Network.Msg,"");
    ret = RxBuffer(i, Network.Msg);
    switch (ret)
    {
        case Busy : break;
        case Ok  : { Network.Status = rxdlist;
                    break;
                }
        case Err  : { Network.Status = disconnect;
                    break;
                }
        default  : break;
    }
    break;
}

case rxdlist : { ret = RxdBuffer(i, &Len);
    rLen.Command = 1;
    switch (ret)
    {
        case Busy : break;
        case Ok  : { if (temp200 == 0)
                    {
                        Network.Status = idle;
                        if (temp400 == 0)
                        {
                            temp400 = 1;
                        }
                        else
                        {
                            tstop = time(NULL);
                            printf("Transmitted file successfully in %.0f
seconds\n",diffime(tstop, tstart));
                        }
                    }
                }
        else
        {
            if (temp200 <= sizeof(Network.Msg))
            {
                tstop = time(NULL);
                gotoxy (3,2);
                printf("Transmitted file in %.0f
seconds\n",diffime(tstop, tstart));
            }
        }
    }
}

```



```

Network.Status = idle;
gotoxy (5,3);
for (blocks = 0;blocks <= 25;blocks++)
{
    putch (219);
}
temp200 = sizeof(Network.Msg);
delay(2000);
close_window (27,10,60,13);
window (1,5,80,22);
break;
}
temp200 = temp200-sizeof(Network.Msg);
gotoxy (5,3);
for (blocks = 0;blocks <= (100-
(temp200/temp201)*100)/4;blocks++)
{
    putch (219);
}
Network.Status = reqlist2;
}
break;
}
case Err :{ Network.Status = disconnect;
break;
}
default : printf("No return value for RxdBuffer\n");
printf("Disconnect on default\n");
Network.Status = disconnect;
break;
}
break;
}

case disconnect :{ close_tcp(arTCP[i].ofsConn);
del_tcp(arTCP[i].ofsConn);
printf("$ Netwerk sessie met versamelstasie verbreek\n");
Network.Status=offline;
break;
}
default : break;
}
}

void WarningSound(void)
{
    sound(4000);
    delay(100);
    nosound();
    delay(10);
    sound(1000);
    nosound();
}

void User()
{
    unsigned k=0,i=0;

```

```
FILE *config_file;

typedef char temp1[10];

struct Net_Names
{
    temp1 Host;
    temp1 Server;
    char temp;
} Net_Names;

for (i=0;i<=10;i++)
{ char init=0;
  Net_Names.Host[i]=init;
  Net_Names.Server[i]=init;
  Net_Names.temp=init;
}

if((config_file = fopen("setup.cfg","r")) == NULL)
{
  exit(1);
}
else
{
  do
  {
    Net_Names.temp = fgetc(config_file);
    switch (Net_Names.temp)
    {
      case EOF: break;
      case '[': k++;
        for (i=0; ;i++)
        {
          if((Net_Names.temp = fgetc(config_file)) == EOF) break;
          if (Net_Names.temp == ']')
          {
            Net_Names.temp="";
            break;
          }
          switch (k)
          {
            case 1: Net_Names.Host[i] = Net_Names.temp;
                     break;
            case 2: Net_Names.Server[i] = Net_Names.temp;
                     break;
          }
        }
        break;
      default : break;
    }
  } while (Net_Names.temp != EOF);
  fclose(config_file);
}

strcpy(MyNetname,Net_Names.Host);
strcpy(Network.Hostname,Net_Names.Server);
}
```

```

void NetInit(void)
{ open_window (15,10,65,13);
  win (15,10,65,13,YELLOW);
  textcolor (BLACK);
  gotoxy (15,2);
  printf("Wag: Inisialiseer netwerk.");
  delay (500);
  switch (bInit)
  {
  case TRUE:
    {
      gotoxy (3,3);
      printf("Waarskuwing ==> Netwerk inisialiserings fout\n");
      delay (500);
      close_window (15,10,65,13);
      window (1,5,80,22);
      break;
    }
  case FALSE:
    {
      gotoxy (11,3);
      printf("Netwerk inisialisering suksesvol\n\n");
      Network.Status=offline;
      delay (500);
      close_window (15,10,65,13);
      window (1,5,80,22);
      break;
    }
  default: break;
  }
}

void Menu(void)
{
  textbackground (MAGENTA);
  textcolor (BLACK);
  clrscr();
  colour=MAGENTA;
  win (1,23,80,25,colour);
  gotoxy (35,2);
  textcolor (BLACK);
  cprintf("TCP/IP Host");
  gotoxy (21,3);
  cprintf(" KOPIEREG: P.S. Veldtsman TECHNIKON OVS ");

  window (1,1,80,25);
  colour=LIGHTGRAY;
  win (1,1,80,4,colour);
  textcolor (BLACK);
  textbackground (LIGHTGRAY);
  gotoxy (34,1);
  cprintf (" COMMANDS ");
  textcolor (WHITE);
  gotoxy (3,2);
  cprintf("<F1> Begin <F2> Eindig <F3> File <F4> <F5>");
  gotoxy (3,3);
  cprintf("<F6> <F7> <F8> <F9> <F10> Quit");
}

```

```

window (1,1,80,25);
textbackground (BLUE);
textcolor (WHITE);
colour=BLUE;
window (1,5,80,22);
clrscr();
}

void API_Loaded(void)
{
void interrupt (*int_apiv)();
int_apiv = getvect(API_INT);
if (int_apiv == 0 || FP_SEG(int_apiv) >= 0xf000)
{
_setcursortype (_NOCURSOR);
win (20,8,60,13,RED);
textcolor (WHITE);
gotoxy (18,1);
cprintf (" FOUT ");
textcolor (BLACK);
gotoxy (9,3);
cprintf ("INET API nie gelaai nie!");
gotoxy (4,4);
cprintf ("Druk 'n sleutel en laai API voordat");
gotoxy (4,5);
cprintf (" die program begin word.");
getch();
exit(1);
}
}

void close_tcp(unsigned tcho)
{
int errcode;

_BX = tcho;
_AH = API_TCPCLOSE;
geninterrupt(API_INT);
if ((errcode = _AX) != 0 && !error)
    sprintf(err_str,"Error on TCP_CLOSE: %s\n",err(errcode));
}

void del_tcp(unsigned tcho)
{
int errcode;

_BX = tcho;
_AH = API_TCPDELET;
geninterrupt(API_INT);
if ((errcode = _AX) != 0 && !error)
    sprintf(err_str,"Error on TCP_DELET: %s\n",err(errcode));
}

char *err(unsigned errcode) /* return errorcode string */
{
static char unk[30];
static char *errs[] = {

```



```

        "No error",
        "Connection already exists",
        "Connection does not exist",
        "Connection closing",
        "No memory for TCB or buffer creation",
        "Would block",
        "Protocol or mode not supported",
        "Invalid arguments"
    };

    if (errcode > INVALID) {
        sprintf(unk, "Unknown error %u", errcode);
        return unk;
    }
    return errs[errcode];
}

/*****
// PROCEDURES FOR WINDOW OPENING AND CLOSING
*****/

void open_window(char x11,char y11,char x22,char y22)
{
    window (x11,y11,x22,y22);
    window_number=window_number+1;
    gettext (x11,y11,x22,y22,window_buffer[window_number]);
}

void close_window(char x11,char y11,char x22,char y22)
{
    puttext (x11,y11,x22,y22,window_buffer[window_number]);
    window_number=window_number-1;
    window (1,1,80,25);
}

void win(int x1,int y1,int x2,int y2,char colour)
{
    textbackground(colour);
    textcolor (WHITE);
    window (x1,y1,x2,y2);
    clrscr();
    x2=x2-x1;
    x1=1;
    y2=y2-y1;
    y1=1;
    block1(x1,y1,x2,y2);
}

void block1(int x1,int y1,int x2,int y2)
{
    int i;

    gotoxy (x1+1,y1);
    putch (left_top);
    for (i=x1;i<x2-2;i++)
        putch (horizontal);
    putch (right_top);
    putch ("\n");
}

```

```

putch ('\r');
for (i=y1+1;i<y2+1;i++)
{
    gotoxy (x1+1,i);
    putch (vertical);
    gotoxy (x2,i);
    putch (vertical);
    putch ('\n');
    putch ('\r');
}
gotoxy (x1+1,y2+1);
putch (left_bottom);
for (i=x1;i<x2-2;i++)
    putch (horizontal);
putch (right_bottom);
x2=x2/2;
y2=y2/2;
gotoxy(x2,y2);
}

/*****
//          Main Procedure
*****/

main(void)
{
    unsigned tcbo,errno,ch,i,acnt;
    int blocks;
    File_Name_Ptr = rLen.File_Name;

    bInit=FALSE;

    Menu();
    API_Loaded();
    User();
    NetInit();
    while(1)
    {
        if (kbhit())
        {
            ch = getch();
            switch (ch)
            {
                case 0: { // function key
                    ch = getch();
                    switch (ch)
                    {
                        case 45: { // alt X
                            if (Network.Status==offline)
                            {
                                close_tcp(arTCP[1].ofsConn);
                                del_tcp(arTCP[1].ofsConn);
                                printf("$ Network sessie met versamelstasie verbreek\n");
                                exit(1);
                            }
                        }
                    }
                }
                case 31: { // alt S

```

```

        status(segTCB,ofsTCB);
        break;
    }

case 59: { //F1
    if (Network.Status==offline)
        Network.Status=LAN;
    break;
}

case 60: { //F2
    if (Network.Status==idle)
        Network.Status=disconnect;
    else
        WarningSound();
    break;
}

case 61: { //F3
    if(Network.Status==idle)
    {
        Network.Status=reqlist1;
        printf("Enter Filename: \n");
        printf("Example: *****.***\n");
        File_Name_Ptr = gets(rLen.File_Name);
        textbackground (BLUE);
        clrscr();
        open_window (27,10,60,13);
        win (27,10,60,13,CYAN);
        _setcursortype (_NOCURSOR);
        gotoxy (5,3);
        for (blocks = 0; blocks <= 25; blocks++)
        {
            putch (177);
        }

    }
    else
        WarningSound();
    break;
}

case 68: { //F10
    if (Network.Status==offline)
        exit(1);
    else
        WarningSound();
    break;
}

default: WarningSound();
        break;
}
}
}
else
{

```

```
TimeSlice();  
    }  
    }  
}
```


BYLAAG 3 - HOSTMOD.CPP

```
/**
 * Modem dialing sequence
 * File: HOSTMOD.CPP
 * Last changed: 11/07/96
 */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include <stdarg.h>
#include <ctype.h>
#include <conio.h>
#include <iostream.h>
#include <sclass.h>
```

```
void Beep_Error(char *c);
void Dial_Error(char *c);
void configuration(void);
```

```
typedef char temp1[10];
```

```
static unsigned Int_Status_Word=0;
```

```
struct Modem_Com
{
    temp1 Tel_Num;
    temp1 Com_Port;
    temp1 Baud;
    temp1 Init_String;
    char temp;
} Modem_Command;
```

```
void configuration()
```

```
{
    unsigned k=0,i=0;
    FILE *config_file;
```

```
for (i=0;i<=10;i++)
{ char init=0;
    Modem_Command.Tel_Num[i]=init;
    Modem_Command.Com_Port[i]=init;
    Modem_Command.Baud[i]=init;
    Modem_Command.Init_String[i]=init;
```

```
}
```

```
if((config_file = fopen("setup.cfg","r")) == NULL)
{
    printf("Cannot open file\n");
    exit(1);
```

```
}  
else  
{  
do  
{  
Modem_Command.temp = fgetc(config_file);  
switch (Modem_Command.temp)  
{  
case EOF: break;  
case '{': k++;  
for (i=0; ;i++)  
{  
if((Modem_Command.temp = fgetc(config_file)) == EOF) break;  
if (Modem_Command.temp == '{')  
{  
Modem_Command.temp=' '  
break;  
}  
switch (k)  
{  
case 1: Modem_Command.Tel_Num[i] = Modem_Command.temp;  
break;  
case 2: Modem_Command.Com_Port[i] = Modem_Command.temp;  
break;  
case 3: Modem_Command.Baud[i] = Modem_Command.temp;  
break;  
case 4: Modem_Command.Init_String[i] = Modem_Command.temp;  
break;  
}  
}  
break;  
default : break;  
}  
} while (Modem_Command.temp != EOF);  
fclose(config_file);  
}  
}
```

```
void Beep_Error(char *c)  
{  
sound(800);  
delay(800);  
nosound();  
printf("%s\n", c);  
delay(1000);  
exit(1);  
}
```

```
void Dial_Error(char *c)  
{  
sound(800);  
delay(800);  
nosound();  
printf("%s\n", c);  
}
```

```
void main(void)  
{
```

```

char buf[80], *term, temp;
int len, port;
long baud;
int Modem_Error;

clrscr();
configuration();

strcpy(buf, Modem_Command.Com_Port);
port = atoi(buf);
printf("Modem on COM %d\n", (port+1));
strcpy(buf, Modem_Command.Baud);
baud = atol(buf);
printf("Baud rate = %d\n", baud);
strcpy(buf, "0");
term = (*buf == '1') ? "\r" : "\r\n";

portSerial portSerial(port, baud, _NONE, 8, 1, term);
if (*portSerial.pubError)
{
    printf(portSerial.pubError);
    exit(1);
}
portSerial.DTR_High();           //DTR and RTS high
delay(1000);
portSerial.CTS_High();          //CTS high
delay(1000);

Int_Status_Word = portSerial.Get_Int_Status();
portSerial.Disable_INT();
printf("Resetting modem\n");
portSerial << "ATZ";             //reset modem
Modem_Error = portSerial.wait_for(191);
if(Modem_Error!=0)
    Beep_Error("Error resetting modem");
else
    printf("Modem reset successfull\n");

printf("Initializing modem\n");
portSerial << strcat("AT",Modem_Command.Init_String); //initialize modem
Modem_Error = portSerial.wait_for(91);
if(Modem_Error!=0)
    Beep_Error("Error initializing modem");
else
    printf("Modem initialized successfull\n");

printf("Dialing %s\n",Modem_Command.Tel_Num);
portSerial << strcat("ATDT",Modem_Command.Tel_Num); //dial telephone number
Modem_Error = portSerial.wait_for(1000);
switch (Modem_Error)
{
    case 0 : printf("OK\n");
              break;
    case 1 : Dial_Error("CONNECT\n");
              portSerial.Enable_INT(Int_Status_Word);
              system("HOST.EXE");
              Int_Status_Word = portSerial.Get_Int_Status();
              portSerial.Disable_INT();

```

```
portSerial.Hang_Up("+++");
portSerial.Enable_INT(Int_Status_Word);
break;
case 2 : break;
case 3 : Dial_Error("NO CARRIER\n");
        exit(1);
        break;
case 4 : Dial_Error("ERROR\n");
        exit(1);
        break;
case 5 : break;
case 6 : Dial_Error("Modem has NO DIALTONE\n");
        exit(1);
        break;
case 7 : Dial_Error("BUSY\n");
        exit(1);
        break;
case 8 : Dial_Error("NO ANSWER\n");
        exit(1);
        break;
default: Beep_Error("Error dialing modem\n");
        exit(1);
        break;
}
}
```


BYLAAG 4 - SERVER.C

```
/**
 * *****
 */
//      TCP/IP SERVER
//      MODULE SERVER.C
//      11/07/96
/**
 * *****
 */

#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include "api.h"

typedef enum flag1 { offline,
                    connect,
                    LAN,
                    disconnect,
                    reqlist,
                    rxlock,
                    idle,
                    connectd,
                    reqdlist,
                    rxlist,
                    rxdlist
                  } teStatus;

typedef struct { char Hostname[79];
               teStatus Status;
               char Msg[500];
               } trNet;

trNet Network;

typedef char tStr80[80];
tStr80 MyNetname;
tStr80 UserName;

void close_tcp(unsigned tcbo);
void del_tcp(unsigned tcbo);
void Init(void);
void TimeSlice(void);
void Write_Buf(void);
void Open_File(void);

/**
 * *****
 */
//      GLOBAL DECLERATIONS
/**
 * *****
 */

#define NIL 0

#define MaxConns 2

/**
 * *****
 */
//      GLOBAL DECLERATIONS TCPBIOS
/**
 * *****
 */

typedef enum TEXFER {elCompl, elBusy, elError, elClosed} teXfer;
```

```

typedef struct TrXfer {
    teXfer eState;
    char *pBuf;
    unsigned cBuf,
    cLen,
    cNow,
    cRdy,
    cErr;
} trXfer;

char kzLenID[14] = "BytesToExpect:";

typedef struct trLen { char zID[15];
    unsigned cLen;
    unsigned J_Value;
    char File_Name[13];
    unsigned Command;
} trLen;

trLen rLen;

void APIAlive(void);
unsigned int TCPOpen(TCP_OP_P op, unsigned far *psegTCB, unsigned far *pofsTCB);
void TxLenData (unsigned segTCP, unsigned ofsTCP, struct TrXfer *ptrNCB);
void RxLenData (unsigned ofsTCP, struct TrXfer *ptrNCB);
void SetLenData (struct TrXfer *ptrNCB, char *ptradbBuf, unsigned cSize);
unsigned TCPPrd(unsigned ofsTCB, struct trLen *pachBuf, unsigned *pcBuf);
unsigned TCPPrd1(unsigned ofsTCB, char *pachBuf, unsigned *pcBuf);
unsigned TCPWr(unsigned ofsTCB, struct trLen far *pachBuf, unsigned *pcBuf);
unsigned TCPWr1(unsigned ofsTCB, char *pachBuf, unsigned *pcBuf);

/*****
//          GLOBAL DECLERATIONS REMTCPB
*****/

const Busy = 0;
const Ok = 1;
const Err = 2;

TCP_OP_P op = {0,0,0x4446,TCP_PASSIVE,0,NIL,NIL,NIL,0,0};

unsigned segTCP, ofsTCP;

enum flag {TRUE,FALSE} bInit;

struct TCPRec { unsigned ofsConn;
    trXfer rTxNCB;
    trXfer rRxNCB;
} arTCP[MaxConns];

TCP_OP_P op;

unsigned ServNetInit(unsigned Index);
enum flag ServConnect(unsigned Index);
unsigned TxBuffer(unsigned Index, char *ptrTxBuf, unsigned TxLen);
unsigned TxdBuffer(unsigned Index);
unsigned RxBuffer(unsigned Index, char *ptrRxBuf);
unsigned RxdBuffer(unsigned Index, unsigned *ptrRxLen);

```

```

//*****
//      PROCEDURES REMTCPB
//*****

```

```

unsigned ServNetInit(unsigned Index)
{
    unsigned cErr;

    bInit = TRUE;
    cErr = TCPOpen(op, &segTCP, &arTCP[Index].ofsConn);
    if (cErr > 0)
        return(2);
    else
        return(0);
}

```

```

enum flag ServConnect(unsigned Index)
{
    TCB far *TCBPtr;
    unsigned cErr;

    APIAlive();
    TCBPtr = MK_FP(segTCP, arTCP[Index].ofsConn);
    switch (TCBPtr -> state)
    {
        case ESTABLISHED:    return (TRUE);
        case LISTEN:        return (FALSE);
        default:             return (FALSE);
    }
}

```

```

unsigned TxBuffer(Index, ptrTxBuf, TxLen)
unsigned Index;
char *ptrTxBuf;
unsigned TxLen;
{
    SetLenData(&arTCP[Index].rTxNCB, ptrTxBuf, TxLen);
    return(Ok);
}

```

```

unsigned TxdBuffer(unsigned Index)
{
    TxLenData(segTCP, arTCP[Index].ofsConn, &arTCP[Index].rTxNCB);
    switch (arTCP[Index].rTxNCB.eState)
    {
        case elBusy:    break;
        case elCompl:  return(Ok);
        default:       return(Err);
    }
    return(Busy);
}

```

```

unsigned RxBuffer(Index, ptrRxBuf)
unsigned Index;
char *ptrRxBuf;
{
    SetLenData(&arTCP[Index].rRxNCB, ptrRxBuf, sizeof(Network.Msg));
}

```

```

return(Ok);
}

unsigned RxdBuffer(Index, ptrRxLen)
unsigned Index;
unsigned *ptrRxLen;
{
  RxLenData(arTCP[Index].ofsConn, &arTCP[Index].rRxNCB);
  *ptrRxLen = arTCP[Index].rRxNCB.cNow;
  switch (arTCP[Index].rRxNCB.eState)
  {
    case eIBusy : break;
    case eICompl: return(Ok);
    default    : { /* Closed, Err */
                  PostTcp();
                  return(Err);
                }
  }
  return(Busy);
}

/*****
//      PROCEDURES TCPBIOS
*****/

unsigned TCPRd(ofsTCB, pachBuf, pcBuf)
unsigned ofsTCB;
struct trLen *pachBuf;
unsigned *pcBuf;
{
  unsigned temp1,temp2;
  _SI = FP_OFF(pachBuf);
  _DX = FP_SEG(pachBuf);
  _CX = *pcBuf;
  _BX = ofsTCB;
  _AH = API_TCPREAD;
  geninterrupt(API_INT);
  temp1 = _CX;
  temp2 = _AX;
  *pcBuf = temp1;
  return(temp2);
}

unsigned TCPRd1(unsigned ofsTCB, char *pachBuf,unsigned *pcBuf)
{
  unsigned temp1,temp2;
  _SI = FP_OFF(pachBuf);
  _DX = FP_SEG(pachBuf);
  _CX = *pcBuf;
  _BX = ofsTCB;
  _AH = API_TCPREAD;
  geninterrupt(API_INT);
  temp1 = _CX;
  temp2 = _AX;
  *pcBuf = temp1;
  return(temp2);
}

```



```

unsigned TCPWr(ofsTCB, pachBuf, pcBuf)
unsigned ofsTCB;
struct trLen far *pachBuf;
unsigned *pcBuf;
{
    unsigned temp1, temp2;
    _SI = FP_OFF(pachBuf);
    _DX = FP_SEG(pachBuf);
    _CX = *pcBuf;
    _BX = ofsTCB;
    _AH = API_TCPWRITE;
    _AL = 0;
    geninterrupt(API_INT);
    temp1 = _CX;
    temp2 = _AX;
    *pcBuf = temp1;
    return (temp2);
}

```

```

unsigned TCPWr1(ofsTCB, pachBuf, pcBuf)
unsigned ofsTCB;
char *pachBuf;
unsigned *pcBuf;
{
    unsigned temp1, temp2;
    _SI = FP_OFF(pachBuf);
    _DX = FP_SEG(pachBuf);
    _CX = *pcBuf;
    _BX = ofsTCB;
    _AH = API_TCPWRITE;
    _AL = 0;
    geninterrupt(API_INT);
    temp1 = _CX;
    temp2 = _AX;
    *pcBuf = temp1;
    return (temp2);
}

```

```

unsigned int TCPOpen(TCP_OP_P op, unsigned far *psegTCB, unsigned far *pofsTCB)
{
    unsigned temp1, temp2, temp3;
    _CX = FP_OFF(&op);
    _BX = FP_SEG(&op);
    _AH = API_TCPOPEN;
    geninterrupt(API_INT);

    temp1 = _BX;
    temp2 = _CX;
    temp3 = _AX;
    *psegTCB = temp1;
    *pofsTCB = temp2;
    return (temp3);
}

```

```

void APIAlive(void)
{
    _AH = API_NOOP;
    geninterrupt(API_INT);
}

```

```

}

void TxLenData(segTCP, ofsTCP, ptrNCB)
unsigned segTCP;
unsigned ofsTCP;
struct TrXfer *ptrNCB;
{
    TCB far *TCBPtr;
    static unsigned cSnd, cTxd;

    if (ptrNCB -> eState == elBusy)
    {
        if (ptrNCB -> cLen == 0)          /* Len not yet sent */
        {
            strcpy (rLen.zID, kzLenID);
            rLen.cLen = ptrNCB -> cBuf;
            cTxd = sizeof(rLen);

            ptrNCB -> cErr = TCPWr(ofsTCP, &rLen, &cTxd);

            switch (ptrNCB -> cErr)
            { case NONE: { if (cTxd == sizeof(rLen))
                { ptrNCB -> cNow = 0;
                  ptrNCB -> cLen = ptrNCB -> cBuf;
                  if (ptrNCB -> cBuf == 0)
                      ptrNCB -> eState = elCompl;
                }
                TCBPtr = MK_FP(segTCP,ofsTCP);
                if (TCBPtr -> state == CLOSED)
                    ptrNCB -> eState = elClosed;
                break;
            }

            case NO_CONN : { ptrNCB -> eState = elClosed;
                            break;
                        }

            case CON_CLOS : { ptrNCB -> eState = elClosed;
                            break;
                        }

            default:      { ptrNCB -> eState = elError;
                          break;
                        }

            }

            if (ptrNCB -> eState != elBusy)
                return;
        }
    }

    if (ptrNCB -> cLen > 0)          /* Len sent */
    {
        cSnd = ptrNCB -> cBuf - ptrNCB -> cNow;
        TCBPtr = MK_FP(segTCP,ofsTCP);
        if (cSnd > TCBPtr -> mss)
            cSnd = TCBPtr -> mss;
        if (cSnd <= ptrNCB -> cRdy)
        {

```



```

CheckPort();
if (ptrNCB -> eState == eIBusy)
{
    if (ptrNCB -> cLen == 0)          /* Len not yet found */
    {
        cRxd = sizeof(trLen);
        /*rNCB.cErr = TcpBIOS.TCPRd(ofsTCP, &rLen, cRxd),*/
        ptrNCB -> cErr = TCPRd(ofsTCP, &rLen, &cRxd);
        switch (ptrNCB -> cErr)
        {
            case NONE: { if (cRxd == sizeof(trLen))
                if (strcmp(rLen.zID, kzLenID) == 0)
                { ptrNCB -> cLen = rLen.cLen;
                    ptrNCB -> cNow = 0;
                    if (ptrNCB -> cLen == 0)
                        ptrNCB -> eState = eICompl;
                    break;
                }
                break;
            }
            case WOULDBLK: { break;
                }
            case NO_SPACE: { break;
                }
            case NO_CONN : { ptrNCB -> eState = eIClosed;
                break;
            }
            case CON_CLOS: { ptrNCB -> eState = eIClosed;
                break;
            }
            default : ptrNCB -> eState = eIError;
                break;
        }
    }

    if (ptrNCB -> eState != eIBusy)
        return;
}

if (ptrNCB -> cLen > 0)          /* Len found */
{
    if (ptrNCB -> cRdy > 0)
    {
        cRxd = ptrNCB -> cLen - ptrNCB -> cNow;
        ptrNCB -> cErr = TCPRd1(ofsTCP, ptrNCB -> pBuf = &Network.Msg[ptrNCB -> cNow],
&cRxd);

        switch (ptrNCB -> cErr)
        {
            case NONE: { ptrNCB -> cNow = ptrNCB -> cNow + cRxd;
                Write_Buf();    // Call routine to write rx buffer to disk
            }
        }
    }
}

```

```

        if (cRxd == 0) /* Remote closed session */
            ptrNCB -> eState = eIClosed;
        else
            if (ptrNCB -> cNow >= ptrNCB -> cLen)
                ptrNCB -> eState = eICompl;
            break;
    }

    case WOULDBLK: break;

    case NO_SPACE: break;

    case NO_CONN: { ptrNCB -> eState = eIClosed;
                    break;
                }

    case CON_CLOS: { ptrNCB -> eState = eIClosed;
                     break;
                 }

    default : ptrNCB -> eState = eIError;
}
}
}
}
}

void Write_Buf(void)
{
    // This routine writes rxbuffer (Network.Msg) to a file;
    static FILE *fp_out;

    if (rLen.Command == 0)
    {
        if ((fp_out = fopen(rLen.File_Name, "wb")) == NULL)
        {
            printf ("Cannot open file\n");
            exit(1);
        }
    }
    if (fwrite(Network.Msg, rLen.J_Value, 1, fp_out) != 1)
    {
        printf ("Error writing to destination file\n");
        fclose (fp_out);
    }

    if (rLen.Command == 2)
    {
        fclose(fp_out);
        exit (1);
    }
}

void Open_File(void)
{
    FILE *fp_out;

    if ((fp_out = fopen(rLen.File_Name, "wb")) == NULL)

```



```

{
    printf ("Cannot open destination file\n");
    exit(1);
}
fclose (fp_out);
}

void TimeSlice(void)
{
    unsigned Len,i;
    enum flag2 {Busy,Ok,Err} ret;

    i=1;

    switch (Network.Status)
    {
        case connect: { if (ServConnect(i) == TRUE)
                        Network.Status = reqlist;
                        break;
                    }

        case reqlist: { ret = TxBuffer(i, Network.Msg, sizeof(Network.Msg));
                        switch (ret)
                        {
                            case Busy : break;
                            case Ok  :{ Network.Status=reqdlist;
                                        break;
                                    }
                            case Err  :{ Network.Status=disconnect;
                                        break;
                                    }
                            default  : break;
                        }
                        break;
                    }

        case reqdlist: { ret=TxdBuffer(i);
                        switch (ret)
                        {
                            case Busy : break;
                            case Ok  :{ Network.Status=rxdlist;
                                        break;
                                    }
                            case Err  :{ Network.Status=disconnect;
                                        break;
                                    }
                            default  : break;
                        }
                        break;
                    }

        case rxlist :{ strcpy (Network.Msg,"");
                        ret=RxBuffer(i, Network.Msg);
                        switch (ret)
                        {
                            case Busy : break;
                            case Ok  :{ Network.Status = rxdlist;
                                        break;
                                    }
                            case Err  :{ Network.Status = disconnect;
                                        break;
                                    }
                        }
                    }
    }
}

```

```

        break;
    }
    default : break;
}
break;
}

case rxclist :{ Len = 499;
    ret=RxdBuffer(i, &Len);
    switch (ret)
    {
        case Busy : break;
        case Ok  :{ Network.Status = reqlist;
            break;
        }
        case Err :{ Network.Status = disconnect;
            break;
        }
        default : break;
    }
    break;
}

case disconnect :{ close_tcp(arTCP[i].ofsConn);
    del_tcp(arTCP[i].ofsConn);
    printf("$ Netwerk sessie met versamelstasie verbreek\n");
    Network.Status=connect;
    break;
}

default : break;
}
}

/*-----*/

void Init(void)
{
    printf("Wag: Inisialiseer netwerk.\n");
    if (ServNetInit(1) != 0)
        printf("Waarskuwing ==> Netwerk inisialiserings fout\n\n");
    else
        printf("Netwerk inisialisering suksesvol\n\n");
    Network.Status=connect;
}

void User()
{
    strcpy(MyNetname,"FANIE");
    strcpy(Network.Hostname,"PIETER");
}

void Menu(void)
{
    printf("Druk <Q> om program te verlaat\n");
}

```

```
void WarningSound(void)
{
    sound(4000);
    delay(100);
    nosound();
    delay(10);
    sound(1000);
    nosound();
}

void close_tcp(unsigned tcb)
{
    int errcode;
    _BX = tcb;
    _AH = API_TCPCLOSE;
    geninterrupt(API_INT);
}

void del_tcp(unsigned tcb)
{
    int errcode;
    _BX = tcb;
    _AH = API_TCPDELET;
    geninterrupt(API_INT);
}

void main(void)
{
    char Kar;

    clrscr();

    User();
    Init();
    Menu();
    while (1)
    {
        if (kbhit())
        {
            Kar = getch();
            if (Kar == 'q')
            { close_tcp(arTCP[1].ofsConn);
              del_tcp(arTCP[1].ofsConn);
              printf("$ Netwerk sessie met versamelstasie verbreek\n");
              exit(1);
            }
        }
        else
            WarningSound();
    }
    else
        TimeSlice();
}
}
```

BYLAAG 5 - SERVMOD.CPP

```
/*
Modem dialing sequence
File: SERVMOD.CPP
Last changed: 11/07/96
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include <stdarg.h>
#include <ctype.h>
#include <conio.h>
#include <iostream.h>
#include <sclass.h>

void Beep_Error(char *c);
void Dial_Error(char *c);
void configuration(void);

static unsigned Int_Status_Word=0;

typedef char temp1[10];

struct Modem_Com
{
    temp1 Tel_Num;
    temp1 Com_Port;
    temp1 Baud;
    temp1 Init_String;
    char temp;
} Modem_Command;

void configuration()
{
    unsigned k=0,i=0;
    FILE *config_file;

    for (i=0;i<=10;i++)
    { char init=0;
      Modem_Command.Tel_Num[i]=init;
      Modem_Command.Com_Port[i]=init;
      Modem_Command.Baud[i]=init;
      Modem_Command.Init_String[i]=init;
    }

    if((config_file = fopen("setup.cfg","r")) == NULL)
    {
        printf("Cannot open file\n");
        exit(1);
    }
}
```

```
else
{
do
{
Modem_Command.temp = fgetc(config_file);
switch (Modem_Command.temp)
{
case EOF: break;
case '{': k++;
for (i=0; ;i++)
{
if((Modem_Command.temp = fgetc(config_file)) == EOF) break;
if (Modem_Command.temp == '{')
{
Modem_Command.temp=' ';
break;
}
switch (k)
{
case 1: Modem_Command.Tel_Num[i] = Modem_Command.temp;
break;
case 2: Modem_Command.Com_Port[i] = Modem_Command.temp;
break;
case 3: Modem_Command.Baud[i] = Modem_Command.temp;
break;
case 4: Modem_Command.Init_String[i] = Modem_Command.temp;
break;
}
}
break;
default : break;
}
} while (Modem_Command.temp != EOF);
fclose(config_file);
}
}
```

```
void Beep_Error(char *c)
{
sound(800);
delay(800);
nosound();
printf("%s\n", c);
delay(1000);
}
```

```
void Dial_Error(char *c)
{
sound(800);
delay(800);
nosound();
printf("%s\n", c);
}
```

```
void main(void)
{
char buf[80], *term, temp;
int len, port;
```



```

long baud;
int Modem_Error;

clrscr();
configuration();

strcpy(buf, Modem_Command.Com_Port);
port = atoi(buf);
printf("Modem on COM %d\n", (port+1));
strcpy(buf, Modem_Command.Baud);
baud = atol(buf);
printf("Baud rate = %d\n", baud);
strcpy(buf, "0");
term = (*buf == '1') ? "r" : "r\n";

portSerial portSerial(port, baud, _NONE, 8, 1, term);
if (*portSerial.pubError)
{
    printf(portSerial.pubError);
    exit(1);
}
portSerial.DTR_High();           //DTR and RTS high
delay(1000);
portSerial.CTS_High();          //CTS high
delay(1000);

Int_Status_Word = portSerial.Get_Int_Status();
portSerial.Disable_INT();

printf("Resetting modem\n");
portSerial << "ATZ";           //reset modem
Modem_Error = portSerial.wait_for(191);
if(Modem_Error!=0)
    Beep_Error("Error resetting modem");
else
    printf("Modem reset successfull\n");

printf("Initializing modem\n");
portSerial << strcat("AT", Modem_Command.Init_String); //initialize modem
Modem_Error = portSerial.wait_for(91);
if(Modem_Error!=0)
    Beep_Error("Error initializing modem");
else
    printf("Modem initialized successfull\n");

LOOP_WEER:

while(!portSerial.charReady())
{

    if(kbhit())
    {
        gets(buf);
        if(toupper(*buf) == 'Q')
        {
            printf("Hanging up modem to quit");
            portSerial.Hang_Up("+++");
            exit(1);
        }
    }
}

```

```
}  
}  
}  
  
portSerial.stringGet(buf, 20, 1000);  
  
if (strstr(buf, "CONNECT"))  
{  
    printf("OK\n");  
    portSerial.Enable_INT(Int_Status_Word);  
    system("SERVER.EXE");  
    Int_Status_Word = portSerial.Get_Int_Status();  
    portSerial.Disable_INT();  
    portSerial.Hang_Up("+++");  
    goto LOOP_WEER;  
}  
  
Beep_Error("Error connecting with host\n");  
exit(1);  
}
```

BYLAAG 6 - COMPARE.C

```
/*
Compare two files
File: COMPARE.C
Last changed: 11/07/96
*/
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    FILE *fp1, *fp2;
    char ch1, ch2, same;
    unsigned long l;

    // See if correct number of command line arguments
    if (argc != 3)
    {
        printf ("Usage: compare <file 1> <file2>\n");
        exit(1);
    }

    // Open first file
    if ((fp1 = fopen(argv[1], "rb")) == NULL)
    {
        printf("Cannot open source file\n");
        exit(1);
    }

    // Open second file
    if ((fp2 = fopen(argv[2], "rb")) == NULL)
    {
        printf("Cannot open file\n");
        exit(1);
    }

    l=0;
    same = 1;

    // compare the files
    while (!feof(fp1))
    {
        ch1 = fgetc(fp1);
        if (ferror(fp1))
        {
            printf ("Error reading first file\n");
            exit(1);
        }

        ch2 = fgetc(fp2);
        if (ferror(fp2))
        {
            printf ("Error reading second file\n");
            exit(1);
        }
    }
}
```

```
if (ch1 != ch2)
{
    printf ("Files differ at byte number %ul",l);
    same = 0;
    break;
}
l++;
}
if (same)
    printf("Files are the same.");

fclose (fp1);
fclose (fp2);
}
```

BYLAAG 7 - KONFIGURASIELEËR

```
//*****
```

```
// Konfigurasielêr vir TCP/IP netwerk sagteware.
```

```
// PS Veldtsman
```

```
// Datum: 07/07/96
```

```
//*****
```

```
{3253} //Telefoon Nommer
```

```
{1} //Com Poort-1
```

```
{9600} //Baud Rate
```

```
{V1Q0X0&K0} //Modem Init String
```

```
[PIETER] //Host Naam
```

```
[FANIE] //Server Naam
```

```
***** Moet geensins aan die formaat van die lêer verander nie *****
```


BYLAAG 8 - API.H

```

/*****
//
//          API.H
//      Hierdie is die globale veranderlikes soos
//      gebruik in programme SERVER.C en HOST.C
/*****

#define API_INT                0x7f

#define API_NOOP                0        /* just keep INET alive */
#define API_SENDDPING           1        /* send an ICMP echo request */
#define API_GETPING             2        /* get an ICMP echo request */
#define API_STARTMAIL           3        /* start the mail sender (client) */
#define API_GETIPADR            4        /* get an IP address, given a name */
#define API_TCPOPEN              5        /* open a TCP connection */
#define API_TCPREAD              6        /* read from a TCP connection */
#define API_TCPWRITE             7        /* write to a TCP connection */
#define API_TCPCLOSE             8        /* close a TCP connection */
#define API_TCPRESET            9        /* reset a TCP connection */
#define API_TCPDELET            10       /* delete TCP connection resources */
#define API_UDPOPEN             11       /* open a udp socket to receive dg's */
#define API_UDPREAD             12       /* read a udp datagram */
#define API_UDPWRITE            13       /* write a udp datagram */
#define API_UDPCLOSE            14       /* close a udp socket */
#define API_GETVARPTR           15       /* get pointer to variables */

/* Error return code */
#define NONE                    0        /* No error */
#define CON_EXISTS              1        /* Connection already exists */
#define NO_CONN                 2        /* Connection does not exist */
#define CON_CLOS                3        /* Connection closing */
#define NO_SPACE                4        /* No memory for TCB creation */
#define WOULDBLK                5        /* Would block */
#define NOPROTO                 6        /* Protocol or mode not supported */
#define INVALID                 7        /* Invalid arguments */
#define BUFSHORT                8        /* Buffer too short for data */
#define RE_ENTER                27       /* Re-entry of Inet */

/* Mode for the API_TCPOPEN call */
#define TCP_PASSIVE              0
#define TCP_ACTIVE              1
#define TCP_SERVER               2        /* Passive, clone on opening */

/* Socket structure */
#ifndef SOCKET_DEF
#define SOCKET_DEF
typedef struct socket {
    long address;                /* IP address */
    unsigned port;              /* port number */
} SOCKET;

/* Connection structure (two sockets) */
typedef struct connection {
    SOCKET local;
    SOCKET remote;
} CONNECTION;

```

```

#endif

#ifndef TIMER_DEF

/* TCP statistics and status variables */
#define NTCB 19
#ifndef TCP_STAT_DEF
#define TCP_STAT_DEF
struct tcp_stat {
    unsigned runt;           /* Smaller than minimum size */
    unsigned checksum;      /* TCP header checksum errors */
    unsigned conout;        /* Outgoing connection attempts */
    unsigned conin;         /* Incoming connection attempts */
    unsigned resets;        /* Resets generated */
    unsigned bdcsts;        /* Broadcast packets received */
    unsigned tcp_window;    /* TCP Maximum offered window */
    unsigned tcp_mss;        /* Maximum segment size to be sent with SYN */
    unsigned tcp_irtt;       /* Initial guess at round trip time */
    unsigned tcp_retry;     /* maximum transmission retries */
    unsigned tel_tos;        /* TOS used by Telnet */
    char flush_flag;        /* To override Nagle */
    struct tcb near *tcbs[NTCB];
};
#endif

struct mem_stat {
    unsigned mem_avail;     /* updated by API_GETVARP call */
    unsigned mem_fail;     /* number of memory alloc failures */
    int stack_avail;        /* lowest stack available observed */
};

/* TCP port numbers */
#define ECHO_PORT 7          /* Echo data port */
#define DISCARD_PORT 9      /* Discard data port */
#define FTPD_PORT 20        /* FTP Data port */
#define FTP_PORT 21         /* FTP Control port */
#define TELNET_PORT 23      /* Telnet port */
#define SMTP_PORT 25        /* Mail port */

typedef struct {           /* UDP write parameter block */
    long rem_ip_ad;        /* remote IP address */
    unsigned rem_port;     /* remote port */
    unsigned length;       /* data count */
    char far *buf;         /* data buffer address */
} UDP_WR_P;

#ifndef TCB_DEF
#define TCB_DEF
typedef struct tcb {      /* TCP Control Block */
    struct tcb near *prev;
    struct tcb near *next;

    CONNECTION conn;

    char state;           /* Connection state */
#define CLOSED 0
#define LISTEN 1

```

```

char tos;                /* Type of service (for IP) */

void near *rcvq;        /* Receive queue */
unsigned rcvcnt;

void near *sndq;        /* Send queue */
unsigned sndcnt;        /* Number of unacknowledged sequence numbers on
                        * send queue. NB: includes SYN and FIN, which don't
                        * actually appear on sndq!
                        */

void near *reseq;       /* Out-of-order segment queue */
TIMER timer;           /* Retransmission timer */
TIMER rtt_timer;       /* Round trip timer */
long rttseq;           /* Sequence number being timed */
long srtt;             /* Smoothed round trip time, milliseconds */
long mdev;            /* Mean deviation, milliseconds */

void near *user;       /* User parameter (e.g., for mapping to an
                        * application control block
                        */

} TCB;
#endif

typedef struct {        /* TCP open parameter block */
    long rem_ip_ad;     /* remote IP address (0 for server) */
    unsigned rem_port;  /* remote port (0 for server) */
    unsigned loc_port;  /* local port (0 for auto assign) */
    unsigned mode;      /* active/passive/server */
    unsigned window;    /* window size (0 for default) */
    /* data receive upcall handler (0 for none) */
    void (far *rcv_upcall)(struct tcb near *tcb,unsigned cnt);
    /* data send upcall handler (0 for none) */
    void (far *snd_upcall)(struct tcb near *tcb,unsigned cnt);
    /* state change upcall handler (0 for none) */
    void (far *stc_upcall)(struct tcb near *tcb,char oldstate,char newstate);
    void near *user;    /* user id field in TCB */
    char tos;          /* type of service to use (normally 0) */
} TCP_OP_P;

```

BYLAAG 9 - SCLASS.H

```

/*****
/*          Serial port class
/*          File: SCLASS.H
/*          Last changed: 11/07/96
*****/

#define FALSE  0
#define TRUE   1

#define _NONE  0
#define _ODD   1
#define _EVEN  2

class portBase
{
public:
    int Address;
    inline void portPut(char c) {outp(Address,c);}
    inline char portGet(void)  {return inp(Address);}
};

class portSerial:portBase
{
private:
    int prvPortnumber;
    long prvBaud;
    int prvParity;
    int prvStop;
    int prvData;
    void setBaud(long);
    void setParity(int);
    void setStop(int);
    void setData(int);
    int systemPortassign(void);
    void systemDelay(unsigned long);

public:
    char *pubTerminator;
    char pubError[80];
    portSerial(int, long, int, int, int, char *);
    ~portSerial(void);
    inline int charReady(void) {return((inp(Address+5)&1)); }
    inline int charGet(void)  {return portGet();}
    int wait_for(long t);
    inline void Disable_INT(void) {outp(Address+1,0x00); }
    inline void Enable_INT(unsigned Status) {outp(Address+1,(inp(Address+1) | Status)); }
    inline unsigned Get_Int_Status(void) {return(inp(Address+1)); }
    inline void DTR_High(void) {outp(Address+4, (inp(Address+4) | 3)); }
    inline void DTR_Low(void) {outp(Address+4, (inp(Address+4) & 0xFC)); }
    inline void DR_Low(void) {outp(Address+5, (inp(Address+5) & 0xFE)); }
    inline void CTS_High(void) {outp(Address+6, (inp(Address+6) | 0x10)); }
    void Hang_Up(char *str);
    int charPut(char);
    int stringGet(char *, int, unsigned long);
    int stringPut(char *);

```



```

inline int operator<<(char c) {return charPut(c); }
inline int operator<<(char *s) {return stringPut(s) ;};
inline void operator>>(char& c) {c=charGet(); }
};

```

```

portSerial::portSerial(int port, long baud, int parity, int data, int stop, char *terminator)

```

```

{
    *pubError = 0;

    pubTerminator = new char[strlen(terminator)+1];
    if(!pubTerminator)
    {
        strcpy(pubError, "Error allocating space for terminating string.");
        return;
    }
    strcpy(pubTerminator, terminator);
    prvPortnumber = port;

```

```

    Address = systemPortassign();

```

```

    if (!Address)
    {
        strcpy(pubError, "Serial port not found.");
        return;
    }

```

```

    setBaud(baud);
    setParity(parity);
    setData(data);
    setStop(stop);
}

```

```

portSerial::~portSerial(void)

```

```

{
    delete pubTerminator;
}

```

```

void portSerial::Hang_Up(char *str)

```

```

{
    char *ptr;

    ptr=str;
    while(*ptr)
        charPut(*ptr++);
    delay(5000);
    ptr="ATH";    //hang-up command
    while(*ptr)
        charPut(*ptr++);
    ptr = pubTerminator;
    while(*ptr)
        charPut(*ptr++);
}

```

```

int portSerial::stringPut(char *str)

```

```

{
    char *ptr;

```



```

ptr=str;
while(*ptr)
    charPut(*ptr++);
ptr = pubTerminator;
while(*ptr)
    charPut(*ptr++);
return (strlen(str)+strlen(pubTerminator));
}

```

```

int portSerial::stringGet(char *response, int len, unsigned long tix)
{
    register int j = 0;
    unsigned long cur, max;

    cur = peek(0x000, 0x46C);
    //peek(0x000, 0x46C, &cur,4);
    max = cur + tix;

    while(j < len && cur < max)
    {
        if(charReady())
            response[j++] = charGet();
        cur = peek(0x000, 0x46C);
        //peek(0x000, 0x46C, &cur,4);
    }
    response[j] = 0;
    return strlen(response);
}

```

```

int portSerial::charPut(char c)
{
    while(TRUE)
    {
        if((inp(Address + 5) & 32) != 0)
        {
            systemDelay(1);
            portPut(c);
            return TRUE;
        }
    }
    return FALSE;
}

```

```

void portSerial::setBaud(long baud)
{
    prvBaud = baud;
    unsigned char brdl, brdh;
    unsigned int brd;
    long vrates[7];

    vrates[0] = 1200L; vrates[1] = 2400L;
    vrates[2] = 4800L; vrates[3] = 9600L;
    vrates[4] = 19200L; vrates[5] = 38400L;
    vrates[6] = 115200L;

    for(brd = 0; brd < 7; brd++)
        if(vrates[brd] == prvBaud)
            brd = 99;
}

```

```

if (brd < 99)
{
    strcpy(pubError, "Unsupported Baud rate.");
    return;
}

brd = 1843200L / (prvBaud * 16L);
brdl = brd;
brdh = brd >> 8;

outp(Address + 3, inp(Address + 3) | 128);
outp(Address + 1, brdh);
outp(Address, brdl);
outp(Address + 3, inp(Address + 3) ^ 128);
}

void portSerial::setParity(int parity)
{
    prvParity=parity;

    outp(Address + 3, inp(Address+3) & 199);
    switch(prvParity)
    {
        case _NONE: break;
        case _ODD : outp(Address + 3, inp(Address + 3) | 8);
                    break;
        case _EVEN: outp(Address + 3, inp(Address + 3) | 24);
                    break;
        default  : strcpy(pubError, "Unsupported parity setting.");
    }
}

void portSerial::setStop(int stop)
{
    prvStop = stop;

    //bit 2 is 0 then 1 stop bit, if bit 2 is 1, 2 stop bits
    switch(prvStop)
    {
        case 1 : outp(Address + 3, inp(Address + 3) & 251);
                break;
        case 2 : outp(Address + 3, inp(Address + 3) | 4);
                break;
        default: strcpy(pubError, "Unsupported stop bit setting.");
    }
}

void portSerial::setData(int data)
{
    prvData = data;

    if(prvData < 5 || prvData > 8)
    {
        strcpy(pubError, "Unsupported data bit setting.");
        return;
    }
    outp(Address + 3, inp(Address + 3) & 252);
}

```

LITERATUURLYS

1. ARRL The ARRL Satellite Anthology. USA: League, 1988.
2. ARRL Handbook for Radio Amateurs. USA: League, 1993.
3. Bellamy, J. Digital Telephony. New York: Wiley, 1991.
4. Black, U. Data links Protocols. Englewood Cliffs: N.J. Prentice-Hall, 1993.
5. Booty, F. Satellite-Based Networking Overview. Electronics. No. 84, December 1994, pp. 64-98.
6. Borland International Inc. Turbo C++ users guide. Scotts Valley CA: Borland International, 1992.
7. Bucher, T.T.N. & Rohde, U. Communications receivers Principles and designs. New York: McGraw-Hill, 1988.
8. Calhoun, G. Digital Cellular Radio. Norwood: Artech House, 1988.
9. Comer, D.E. & Stevens, D.L. Internetworking with TCP/IP Volume II Design, Implementation, and Internals. Englewood Cliffs: Prentice-Hall, 1991.
10. Davidson, J. An Introduction to TCP/IP. New York: Springer-Verlag, 1988.
11. Dodds, R. en Grobbelaar, P. Rekenaar woordeboek. Parow: HAUM, 1987.
12. Drake, J. How a GSM Cellular Network works: Which Business Magazine, Vol. 11, No. 3, 1994.
13. Duntemann, J. & Weiskamp, K. PC Techniques C/C++ Power Tools. New York: Bantam Books, 1992.
14. Feit, S. TCP/IP Architecture, Protocols and Implementation. New York: McGraw-Hill, 1993.
15. Halshall, F. Data Communication, Computer Networks and Open Systems. Wokingham: Adison-Wesley, 1992.
16. Hedrick, C.L. Introduction to the Internet Networking Protocols (TCP/IP). New Jersey: Rutgers University, 1987.
17. Higgins B. Cellphones Explained: Elektor Electronics. Vol. 18, No. 201, June 1992, pp. 62-63.

18. Holmes M. & Flanders B. PC Magazine C++ C Communications Utilities. California: Ziff-David Press, 1993.
19. Horzepa, S. Your gateway to packet radio. USA: The League, 1989.
20. Hughes, L. Data Communications. USA: McGraw-Hill Inc, 1992.
21. Kanefsky, M. Communication Techniques for Digital and Analog Signals. New York: Wiley, 1987.
22. Karecki, P.W. Wireless Technology Cuts the Strings: Laptop Buyer's Guide and Handbook. Vol. 12, No. 3, March 1994, pp. 38-51.
23. Kritzinger, M.S.B. & Steyn & Schoonees P.C. & Cronje U.J. Groot woordeboek. Pretoria: JL van Schaick, 1972.
24. Lee, W.C.Y. Mobile Cellular Telecommunications Systems. New York: Mc-Graw-Hill Book Company, 1989.
25. Lui, M. & Papantoni-Kazakos, P. A Protocol for Cellular Radio Signaling Channels Carrying Data and High-Priority Accessing Requests: IEEE Transactions on Communications. Vol. 41, No. 4, April 1993, pp. 570-582.
26. Mannes, G. The Changing Face of Cellular Phones: Popular Mechanics. May 1994, pp. 53-55,117.
27. Microsoft. Microsoft Encarta '95, 1995.
28. Mc Ardle, B.P. Coding for GSM: Elektor electronics. Vol. 20, No 218, January 1994, pp. 46-50.
29. Melville, S. & Goddard, W. Research Methodology. Kenwyn: Juta & Co Ltd, 1996.
30. Minoli, D. Telecommunications Technology Handbook. New Jersey: Prentice-Hall International, 1991.
31. Noll, A.M. Introduction to Telephones and Telephone Systems. Boston: Artech House, 1991.
32. Norton, P. The Peter Norton Programmer's Guide to the IBM PC. Washington: Microsoft Press, 1985.
33. Pritchard, W.L. & Suyderhoud, H.G. & Nelson, R.A. Sattellite Communication Systems Engineering. New Jersey: Prentice-Hall, Inc, 1993.

34. Provinsiale Administrasie van die Vrystaat: Direkoraat Gesondheid en Welsyn.
35. Reconstruction and development programme, Sixth Draft 13 Jan 1995.
36. Ritchie, W.K. & Stern, J.R. Telecommunications Local Networks. London: Chapman and Hall, 1993.
37. Ross, D.T. A C++ Serial Port Class: Microcomputer Journal. Vol. 2, No. 4, July/August 1995, pp. 21-25.
38. Schildt, H. Teach Yourself C. California: Osborne McGraw-Hill, 1990.
39. Schwaderer, W.D. C Programmer's Guide to NetBIOS, IPX and SPX. USA: SAMS, 1992.
40. Stark H. & Tuteur F.B. & Anderson J.B. Modern Electrical Communications. New Jersey: Prentice Hall, 1988.