Ingeniørhøjskolen Aarhus – Aarhus Universitet

Gruppemedlemmer

Jacob L. F. Kurtzhals – AU537301, 201505507  
Rasmus E. Z. Appel – AU547553, 201506670  
Rasmus Lund – AU264089, 201310517  
Joachim Lenbroch – AU470780, 06335

øvelse 12 journal Protokolstak

E17IKN – Gruppe 42

Indhold

[Udviklingsfoløb & Funktionalitet 2](#_Toc500849631)

[Link Layer 2](#_Toc500849632)

[Transport Layer 2](#_Toc500849633)

[Application Layer 2](#_Toc500849634)

[File Client 2](#_Toc500849635)

[File Server 2](#_Toc500849636)

[Resultater 2](#_Toc500849637)

# Udviklingsfoløb & Funktionalitet

Her kan læses om vores udviklingsforløb af de forskellige lag i øvelsen, samt kode snippets med dertilhørende forklaringer.

## Link Layer

Lorem ipsum

## Transport Layer

I dette lag skulle implementeres en stop-and-wait protokol, denne skulle være pålidelig og anvende en 16-bit checksum, og kunne håndtere ødelagte pakker. Laget overholder en given headerformat, som kan ses nedenfor. Til udviklingen af dette lag er der allerede udviklet metoder til at sende og modtage ACKs , og udviklet en klasse til at udregne og verificere checksums.

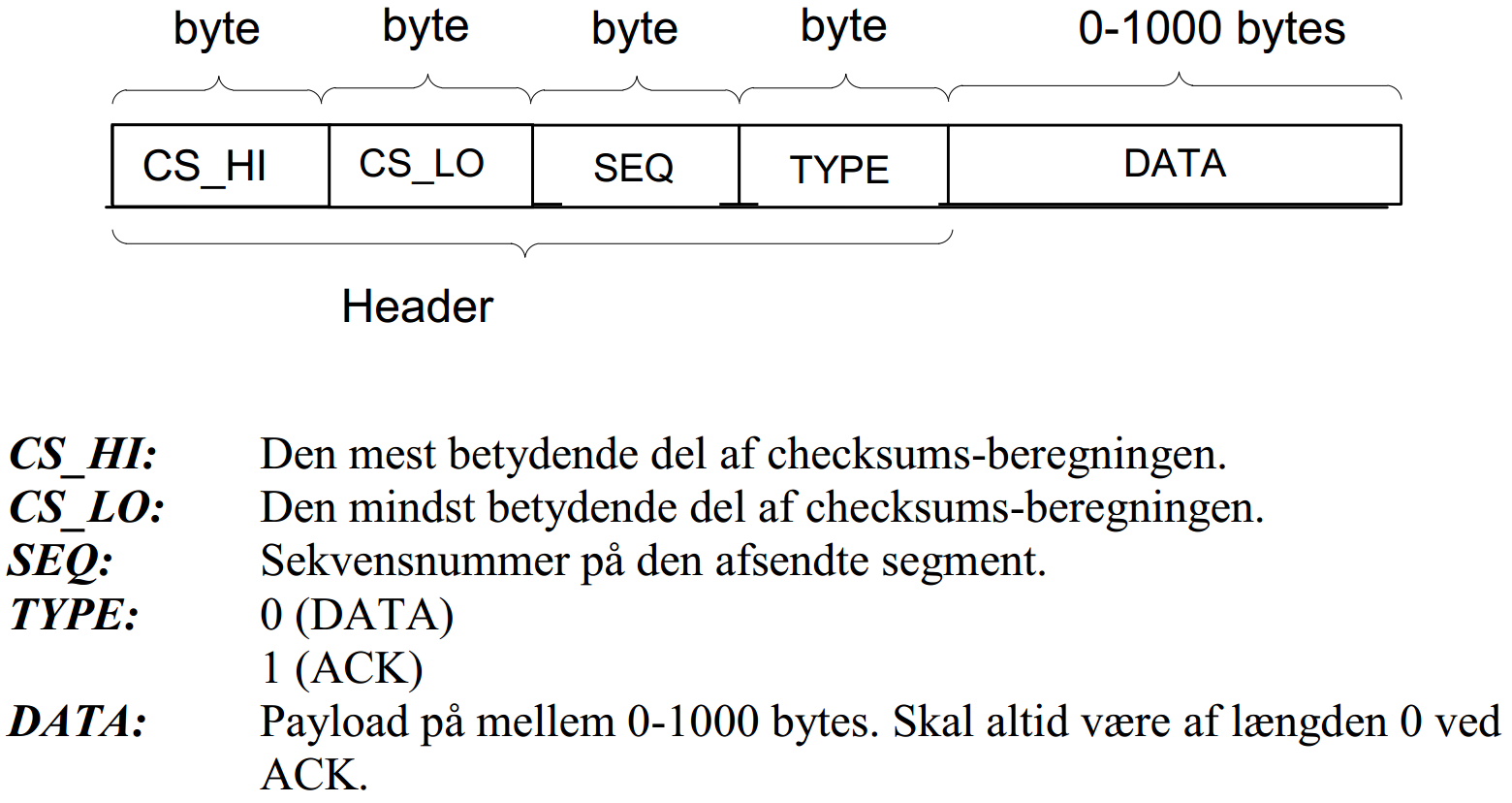


Figure 1, sgment header format

### Send

Send metoden har formen *send(byte[] buf, int size)* , som returnerer void. Denne modtager et array, og en størrelse på dette, derfor er størrelsen på pakkerne der sendes styret af applikationslaget. Metoden står for at tildele en header til den givne *buf*, og derefter sende denne pakke via link laget.

var sendBuf = new byte[size + 4];

sendBuf[(int)TransCHKSUM.SEQNO] = seqNo;

sendBuf[(int)TransCHKSUM.TYPE] = (byte)TransType.DATA;

Array.Copy(buf, 0, sendBuf, 4, buf.Length);

checksum.calcChecksum(ref sendBuf, sendBuf.Length);

Ovenfor ses et kodeudsnit fra starten af metoden, der oprettes et array ud fra metodens parametre plus de fire header bytes, dette sikrer at det array der sendes har den ønskede størrelse. Herefter sættes sekvensnummeret til det nuværende, og typen sættes til 0 (DATA). Herefter lægges buf over i sendBuf, og der beregnes checksum.

Herefter startes en do-while løkke, hvor den oprettede sendBuf sendes, herefter modtages en ACK vha. receiveAck() metoden, hvis denne returnerer falsk, inkrementeres sendErros med en og pakken forsøges sendt igen.

bool receiveAckBool;

var sendErrors = 0;

do

{

link.send(sendBuf, sendBuf.Length);

receiveAckBool = receiveAck();

if (!receiveAckBool)

{

sendErrors++;

if (sendErrors > 5)

{

throw new System.Exception("SendTimeOutException");

}

}

} while (!receiveAckBool);  
 old\_seqNo = DEFAULT\_SEQNO;

En vigtig del af denne metode er at old\_seqNo sættes til DEFAULT\_SEQNO, dette skal gøres for at nulstille sekvensnummeret efter send metoden er udført, og gør det muligt at starte en ny modtagelsessekvens.

### Receive

Denne metode har formen receive(ref byte[] buf), dette vil sige at der modtages en reference til et array, som redigeres i metoden. Denne metode har til formål at modtage en pakke fra linklaget, og tjekke at checksum og sekvensnummer er korrekt, og ellers håndtere hvis de ikke er.

do

{

Array.Clear(buffer, 0, buffer.Length);

recvSize = link.receive(ref buffer) - 4;

//Insert transmissionnoise here

var recErrors = 0;

while (!checksum.checkChecksum(buffer, buffer.Length))

{

recErrors++;

if (recErrors > 5)

{

throw new System.Exception("ReceiveTimeOutException");

}

sendAck(false);

Array.Clear(buffer, 0, buffer.Length);

recvSize = link.receive(ref buffer) - 4;

}

if (buffer[(int)TransCHKSUM.SEQNO] == old\_seqNo)

{

sendAck(true);

}

} while (buffer[(int)TransCHKSUM.SEQNO] == old\_seqNo

Metoden starter ud med en do-while løkke, som rydder bufferen og modtager en pakke igennem linklaget, returværdien fra denne trækkes der fire fra, således at størrelsen kun er størrelsen på datadelen af pakken. Herefter tjekkes checksummen, hvis denne ikke er sand sendes en false ACK, og der forsøges modtaget en ny pakke, dette forsøges et antal gange før en exception bliver thrown. Efter checksum er godkendt tjekkes om det nuværende sekvensnummer er lig det gamle, hvis det er, sendes en sand ACK og løkken starter forfra. Der kan desuden indsættes kode til at simulere transmissionsfejl før checksummen tjekkes.

old\_seqNo = buffer[(int)TransCHKSUM.SEQNO]; //update old seqno to newest one

sendAck(true);

Array.Copy(buffer, 4, buf, 0, buffer.Length - 4);

return recvSize;

Efter løkken sættes old\_seqNo til den nye, der sendes en sand ACK, og data fra buffer kopieres til buf, til sidst returneres en receiveSize.

## Application Layer

### File Client

Filklienten ligner meget den der blev udviklet I den sidste obligatoriske opgave, altså der sendes en filsti til server, som så sender en filstørrelse tilbage, ud fra denne filstørrelse startes enten modtagelse af en fil, eller programmet afsluttes.

var receiveBuffer = new byte[BUFSIZE];

var receivedData = new byte[] { };

var receiveSize = 0;

int index = 0;

do

{

receiveSize = transport.receive(ref receiveBuffer);

Array.Resize(ref receivedData, index + receiveSize);

Array.Copy(receiveBuffer, 0, receivedData, index, receiveSize);

index += receiveSize;

### } while (receivedData.Length != fileSize); //Keeps looping until the desired filesize is reached

Ovenfor ses den kodestump der er ansvarlig for modtagelse af filen, først oprettes et array til at opbevare det modtagede data, med en givet størrelse (i dette tilfælde 1000), det næste array der oprettes bruges til at opbevare al den modtagede data, og er derfor oprettet uden størrelse. receiveSizes opgave er at holde styr på den nuværende modtagede størrelse, og index har ansvar for at holde styr på hvor der skal kopieres til i receivedData.

I do-while løkken modtages en pakke og en filstørrelse fra transportlaget, herefter ændres størrelsen på receivedData ud fra det nuværende index og den modtagede filstørrelse. Til sidst kopieres den modtagede data over i receivedData, og index forøges. Dette gentages indtil størrelsen på receivedData er lig den forventede filstørrelse.

## File Server

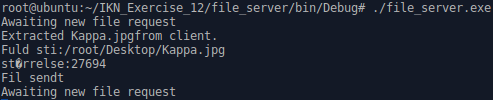
Lorem ipsum

# Resultater

Lorem ipsum

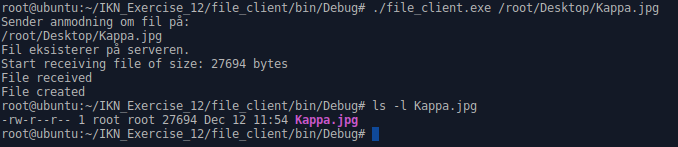
## Uden fejl

Lorem ipsum



Figur 1 - Sending af fil fra File\_server.

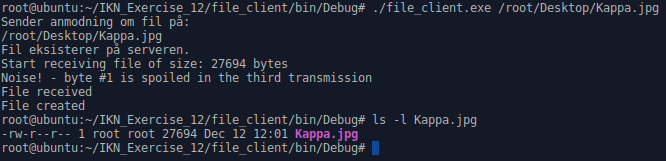
Lorem ipsum



Figur 2 - Modtagelse af fil på File\_client.

## Bitfejl I data pakke

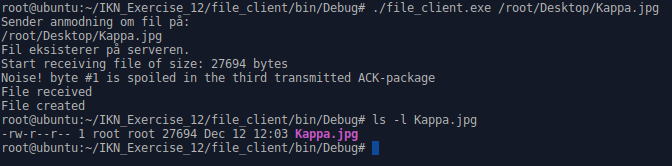
Lorem ipsum



Figur 3 - Tvang af bitfejl i data pakke.

## Bitfejl I ACK pakke

Lorem ipsum



Figur 4 - Tvang af bitfejl i ACK pakke.