Tidmätningsnoggrannhet i datanät

Marko Björninen

Examensarbete

Informationsteknik

2015

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	13018
Författare:	Marko Björninen
Arbetets namn:	Tidmätningsnoggrannhet i datanät
Handledare (Arcada):	Göran Pulkkis
Uppdragsgivare:	Jonny Karlsson

Sammandrag:

Examensarbetets syfte är att utreda hur noggranna tidmätningar det går att utföra i datanät. Noggranna tidmätningar kan användas till olika syften, som t.ex. avståndsuppskattning. Målet var att skapa tidsstämplar med en resolution bättre än tio nanosekunder. Denna resolution skulle i avståndsuppskattning motsvara ungefär tre meters noggrannhet då ett datapaket kan antas flyga med ljusets hastighet. Avhandlingen är en praktisk tillämpning av tidigare forskning. Fokus är på mjukvara som kunde skapa tillräckligt noggranna tidsstämplar. Som programmeringsspråk används både C och Assembler. Mjukvaran mäter processeringstid, överföringstid och flygtid med skapade tidsstämplar. Mätresultatet som uppnås i processeringstidmätningen är tillräckligt noggrant för tillförlitlig uppskattning av overheadtiden. I överförings- och flygtidsexperimenten är mätresultatens varians för stor för tillförlitlig uppskattning av ett datapakets överförings- och flygtid. Slutsatsen som nås är att det inte går att med mjukvarutidsstämplar tillförlitligt uppskatta ett datapakets flygtid med tillräcklig resolution. Det krävs alltså hårdvarustöd för att tillförlitligt uppnå en tillräcklig resolution för användning av mätresultatet t.ex. vid avståndsuppskattning.

Nyckelord:	Tidmätning, flygtid, processeringstid, exekveringstid, överföringstid
Sidantal:	43
Språk:	Svenska
Datum för godkännande:	18.7.2016

DEGREE THESIS	
Arcada	
Degree Programme:	Information Technology
Identification number:	13018
Author:	Marko Björninen
Title:	Accuracy of Time Measurements in Networks
Supervisor (Arcada):	Göran Pulkkis
Commissioned by:	Jonny Karlsson

Abstract:
The goal of this thesis is to investigate the precision of timestamps created in networks. Timestamps are useful for many different purposes with localization being the most common. Based on the knowledge that a datagram travels through air at the speed of light the aim is to create timestamps with a resolution of at least ten nanoseconds. This resolution would equal a three-meter accuracy for localization purposes. C and Assembly language programs have been created for measuring processing time in a computer, transfer time of a data packet between two computers in a network, and the flight time of a data packet between two computers in a wireless ad hoc network. The results achieved in the processing time measurements were accurate enough for determination of the overhead time. The conclusion is that a credible estimate of the flight time of a data packet is impossible to obtain with software timestamps. Hardware components are required for reliable creation of precise timestamps, for example for localization purposes.

Keywords:	Time measurement, time of flight, processing time, transfer time
Number of pages:	43
Language:	Swedish
Date of acceptance:	18.7.2016

Innehåll

Figurer

Tabeller

Terminologi och förkortningar

API = Application Programming Interface, applikationsprogrammeringsgränssnitt
FIFO = First In First Out, kö-struktur där först anlända betjänas först
GPS = Global Positioning System, globalt lokaliseringssystem
MANET = Mobile Ad hoc NETwork, mobilt infrastukturlöst nätverk
Overhead = Tiden mellan det att kommandot anropats och tills det exekveras
Processeringstid/Processingtime, PT = Tiden för att exekvera ett eller flera instruktioner
RDTSC = Read Time Stamp Counter, assemblerinstruktion för att avläsa processorns tidstämpel
Tidsstämpel = En tidpunkt som skapas och sparas då ett specifikt kommando körs
TTHCA = Traversal Time and Hop Count Analysis
UDP = User Datagram Protocol, ett protokoll för att skicka datapaket i datanät

1 INLEDNING

1.1 Tidmätningsnoggrannhet

I denna avhandling kommer jag att utreda hur noggrant det går att mäta tider i datanät. Med noggranna och tillförlitliga tidsstämplar kan man räkna ut datapakets flygtider och processeringstider, vilket har en betydelse i vissa algoritmer såsom t.ex. TTHCA (Karlsson, Dooley, & Pulkkis, 2011). Datapakets flygtider kan även användas t.ex. till att uppskatta distansen mellan två noder i ett trådlöst nätverk genom att räkna med att datapaketet flyger med ljusets hastighet. Uppskattningen av distans blir mera invecklad då det förekommer fysiska barriärer och kan då sannolikt bli otillförlitlig.

En dators hårdvara har stor inverkan på tidmätningsnoggrannheten, t.ex. processorns klockfrekvens och arkitektur. Datorernas nätverkskort spelar också en viktig roll i att skapa noggranna tidsstämplar för att uppnå noggranna tidmätningar.

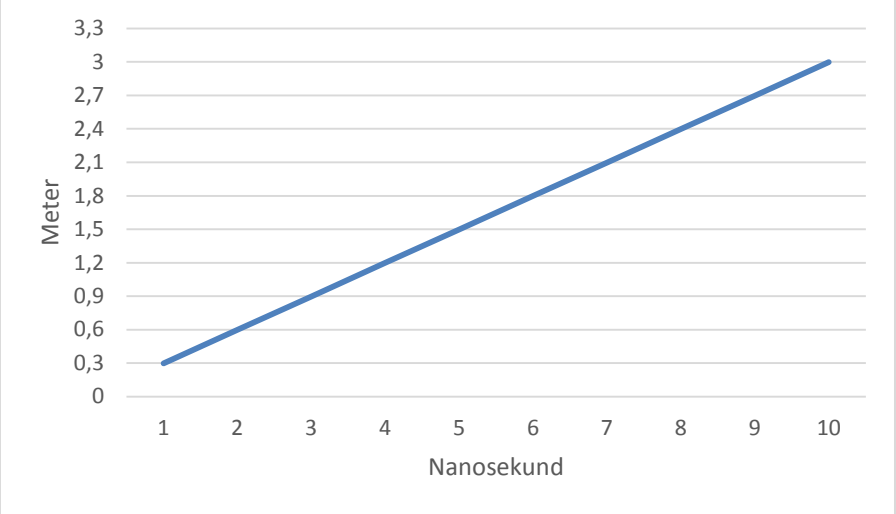
1.2 Metod

Avhandlingen är en praktisk tillämpning av tidigare forskning. Dess syfte är att undersöka hur noggrant man kan skapa tidsstämplar i datanät. För undersökningen används både färdiga program och bibliotek som utlovar tidsstämplar med den noggrannhet som krävs för undersökningens mål, dvs. $\leq$ 10 nanosekunder. Dessutom redogörs hur man uppnår noggranna tidmätningar med hårdvarukomponenter, som är specificerade för tidsstämplar med minst 10 nanosekunders resolution. Därtill nyttjas dokumentationen av sådana hårdvarukomponenter.

1.3 Målsättning

Resolutionen som eftersträvas för noggranna tidsstämplar är $\leq$ 10 nanosekunder. Ljusets hastighet är $3*10^8$ meter per sekund vilket innebär att ljuset rör sig cirka 0,3 meter per nanosekund (se Fig. 1). Om möjligt så är noggrannare tidsstämplar naturligtvis önskvärda. Med tidsstämplar som uppnår den målsatta resolutionen kan man t.ex.

beräkna distansen mellan två datorer i ett nätverk med 3 meters noggrannhet. Denna resolution är också tillräcklig t.ex. för TTHCA-algoritmens (Karlsson, Dooley, & Pulkkis, 2011) behov.



Figur 1. Signalutbredning med ljusets hastighet.

2 BAKGRUND

2.1 Datanät

Flygtidmätningen fungerar både trådlöst och över en trådbaserad förbindelse. I de trådlösa nätverken där noderna har fri sikt är flygtiderna kortast. Ifall noderna inte har fri sikt blir flygtiden längre. Likväl blir överföringstiden längre då datapaket skickas över en trådbaserad förbindelse. Paketets överföringstid över en trådbaserad förbindelse blir svår att uppskatta ifall man inte vet kabelns längd och typ, eller om paketet överförs t.ex. via en switch.

2.1.1 Trådlöst datanät

I trådlösa datanät kan man utföra avståndsmätningar med ungefär tre meters noggrannhet när datapaketens flygtid kan mätas med 10 nanosekunders resolution. Detta görs genom att räkna datapaketets flygtid gånger ljusets hastighet. Då noderna har fri sikt mellan varandra är resultatet mycket pålitligt. Ifall det finns fysiska barriärer

mellan noderna kan mätresultatet och avståndsestimeringen bli otillförlitliga.

2.1.2 Trådbaserade nätverk

I trådbaserade nätverk fungerar tidmätning lika väl som i trådlösa nätverk. Distansuppskattning med överföringstidmätning är svår att utföra i trådbaserade nätverk av tre huvudsakliga orsaker. Första orsaken är att det kan vara svårt ifall inte omöjligt att uppskatta distansen mellan datorerna ifall kablarna inte är synliga hela vägen. Andra orsaken är att ett datapaket färdas med olika hastighet beroende på kabelns material. Tredje orsaken är att datapaket kan färdas genom switchar, routrar och brandmurar innan det når destinationsdatorn.

2.2 Tidmätningsbehov

Tidmätning är något som förekommer ofta i vardagen, t.ex. i form av tidmätning på millisekundnivå i löptävlingar eller på sekundnivå i en mikrovågsugn. Inom datatekniken är resolutionskravet ofta mycket högre, dvs. tidsstämplarna måste ofta vara noggrannare än en mikrosekund. De vanligaste användningsområdena för tidmätning inom datatekniken är avståndsuppskattning, datasäkerhet och optimeringssyften. För avståndsuppskattning krävs en resolution som är noggrannare än 10 nanosekunder.

2.3 Tidpunkter i datanät

Med tanke på användningsområdena är de relevanta tidpunkterna i datanätet då en enhet tar emot ett datapaket och då enheten skickar iväg det. Med hjälp av tidsstämplarna kan man då räkna ut processerings- och flygtiderna. För att processerings- och flygtidsräkningarna skall vara tillförlitliga bör dessa tidsstämplar skapas med tillräckligt noggrann resolution och låg varians.

3 RELATERAD FORSKNING

I detta kapitel skriver jag om relaterad forskning, dvs. forskning med liknande syfte eller forskning som baserar sig på noggrann tidsstämpling men inte har haft det som huvudsyfte.

3.1 Distansuppskattning baserat på flygtiden

Noggrann lokalisering har många användningsområden både kommersiellt och icke-kommersiellt. Lokalisering utomhus är kommersiellt tillgänglig och t.ex. GPS används i vardagen av en markant andel av befolkningen i västvärlden. Lokalisering inomhus är inte lika långt utvecklad och inte lika tillgänglig i den öppna marknaden. GPS ger inte tillförlitliga resultat då apparaten är bakom barriärer såsom t.ex. ett plåttak som dessutom reflekterar signaler.

Wibowo et al. (2009) har undersökt hur noggrant man kan uppskatta distansen mellan två noder baserat på datapaketets flygtid. De använde sig av hårdvara som var tillgänglig på öppna marknaden. Formeln de använde sig av är baserad på ljusets hastighet. Experimentet skedde i miljöer där enheterna hade fri sikt. Experimentet började med enheterna bredvid varandra, på noll meters avstånd. Gradvis ökades avståndet upp till 30 och 40 meter. Tre mätningar utfördes i experimentet.

4 TIDMÄTNINGSTEKNOLOGI

Det finns flera olika teknologier som man kan använda för att skapa noggranna tidsstämplar. Tidsstämplarna kan skapas med hjälp av programvara, hårdvara eller en kombination. I detta kapitel kommer jag att gå igenom olika alternativ som kan användas för att skapa noggranna tidsstämplar i datanät.

4.1 Programvara

För att man tillförlitligt ska kunna skapa noggranna tidsstämplar måste det ske på maskinspråknivå, s.k. lågnivå. Av denna orsak är programvarulösningarna få, men jag skall ändå utreda huruvida programvara är en användbar och tillförlitlig tidsmätningsteknologi för noggranna tidsstämplar. Det vanligaste mjukvarulösningarna för analys av nätverkstrafik bygger på libpcap- och WinPcap-biblioteken i Linux- och Windows-miljöer. Med programmeringsspråket C är det även möjligt att med inbyggda funktioner i Linux-datorer skapa tidsstämplar med resolutionen en nanosekund.

4.1.1 Wireshark

Wireshark är ett program som lyssnar på och analyserar nätverkstrafiken via nätverkskort. Wireshark skapar inte noggranna tidsstämplar, men kan registrera tidsstämplar som skapas med WinPcap och libpcap i Windows- respektive Linux-versionerna av Wireshark. (2016)

4.1.2 SharpPcap för C#

SharpPcap är ett ramverk för programmeringsmiljön .NET, som baserar sig på Winpcap- och libpcap-biblioteken. SharpPcap-ramverket möjliggör tilläggsfunktionalitet i hanteringen av datapaket i form av en API. Med hjälp av SharpPcap kan man analysera all trafik som passerar via ett nätverkskort. (Gal & Morgan, 2014)

Syftet med SharpPcap är att erbjuda ett ramverk för att analysera nätverkstrafik för .NET-applikationer i Windowsmiljö. SharpPcap är även tillgängligt i Linux genom projektet (Mono, 2016). I sig själv möjliggör SharpPcap inte tillräckligt noggranna tidsstämplar, men med hårdvarustöd kunde detta ramverk användas för noggranna flygtidmätningar.

4.1.3 C programmering i Linux

I programmeringspråket C finns det funktioner som hämtar datorns epoch-tidstämpel med en nanosekunds noggrannhet (Die.net, 2016). Det som bör tas i beaktande när man programmerar på högre nivå är att processeringstiderna är längre vilket ger tidsstämplarna en overhead som bör utredas för noggranna mätresultat. Ifall man programmerar i C kan dock t.ex. funktionen *clock_gettime* användas för att skapa en vägledande tidsstämpel.

I min tillämpning av BinaryTides (2012) exempel har jag skapat två C – program som båda använder sig av funktionen *clock_gettime* (Bilaga 1 och 2). Båda programmen använder tidstämplingsfunktionen på samma sätt. Funktionen skapar endast nya variabler. Tidstämpeln skrivs in i variabeln och värdet returneras. Redan i denna avskalade funktion kan man klart notera hur stor inverkan olika instruktioner har på processeringstiden. Funktionen som anropar *clock_gettime* visas i sin helhet nedan.

```
long timestamp()
{
	//clock_* variables
	 struct timespec t1, res;
	 int id = CLOCK_REALTIME;

	//get timestamps with nanosecond resolution
	 clock_getres(id, &res);
	 clock_gettime(id, &t1);

	return t1.tv_nsec;
}
```

Dessa program skickar datapaket sinsemellan. Det första programmet (receiver.c) avläser en tidsstämpel, packar in den i ett UDP-datapaket som skickas till mottagaren. Det andra programmet lyssnar på trafiken till en specificerad port. Då ett datapaket anländer till porten så avläser programmet en ny tidsstämpel, packar in det i ett UDP-datapaket och svarar till samma adress. Programmens utmatning:

Timestamp sent: tidstämpel 1
Timestamp received: tidstämpel 2

Dessa program kan användas som riktgivande testprogram. Även om tidsstämpeln som funktionen *clock_gettime* returnerar har resolutionen en nanosekund går det inte att

uppskatta processeringstiderna tillräckligt noggrant eftersom variansen är för hög för att kunna utreda exekveringens overhead. Tiden mellan tidpunkten då ett datapaket tas emot och tidpunkten för en tidsstämpel kan variera med hundratals nanosekunder från gång till gång.

4.1.4 Assembler

I C-programmet jag skapade var processeringstiderna ett problem för tidmätningens noggrannhet. Genom att implementera lågnivåspråk i form av Inline assemblerinstruktioner i C-programmet, kan man förbättra processeringstidernas mätningsnoggrannhet. Dessutom utnyttjas Allan Cruses (2009) assemblerprogram, se Bilaga 5 och 6. I dessa assemblerprogram används RDTSC-instruktionen för att skapa tidsstämplar. Assemblerprogrammen är strukturerade såsom C-programmen i att det finns en avsändare och en mottagare vilka skickar datapaket sinsemellan och skapar tidsstämplar vid ankomst och avsändning.

Det bör beaktas att moderna processorer är mångprogrammerade. Detta innebär att ett program kan avbrytas mitt i exekveringen för processbyte, vilket förlänger processeringstiden och därigenom förvränger tidmätningen. För att säkerställa att ett program inte får avbrytas då det exekveras bör man exekvera det dedikerat på en prosessorkärna samt förbjuda avbrott på denna kärna. Detta kräver då att vårt program skall anropa en kernelmodul som förbjuder avbrott. Avbrott förbjuds i början av programmet, före processeringstidens mätning börjar och avbrott tillåts igen efter att mätningen har utförts. Exekvering av ett enkelt aktiveringsprogram i bakgrunden har samma effekt.

Exekvering av ett program dedikerat på en specifik prosessorkärna inställs med *taskset*-kommandot i Linux.

Ytterligare bör beaktas att moderna processorer sällan jobbar med en konstant klockfrekvens. På grund av detta måste man låsa klockfrekvensen på ett specifikt värde. Detta görs enklast genom att säkerställa att processorn jobbar konstant med maximal

klockfrekvens. Alternativt måste programmet ta reda på klockfrekvensens värde före tidsstämplingarna utförs.

Processorns klockfrekvens

För att säkerställa att processorn jobbar på maximal klockfrekvens har jag stängt av energisparläget. Efter att energisparläget är avstängt exekverar jag ett s.k. aktiveringsprogram, som inte behöver göra något specifikt. Huvudsaken är att det exekveras i bakgrunden med låg prioritet. I mätexperimenten har jag använt mig av ett aktiveringsprogram som exekveras ett obegränsat antal gånger, se Bilaga 4.

Dessa två steg har enligt mina test visat sig vara tillräckliga för att ställa in processorn på den maximala klockfrekvensen. Processorerna på datorerna som används för mätexperimentet har då konstanta klockfrekvenser med värdet ungefär 2,6 GHz.

4.2 Hårdvara

Majoriteten av lösningarna för att skapa noggranna tidsstämplar med en nanosekunds resolution är hårdvarulösningar. Hårdvaran kan inkludera egen programvara för att avläsa tidsstämplar, t.ex. i form av ett nätverkskort med tillhörande programvara.

4.2.1 Napatech nätverkskort

Napatech är en nätverkskortstillverkare som levererar produkter som uppges skapa tidsstämplar med 1 nanosekunds resolution för nätverkstrafiken (Napatech, 2015). Färdiga program som t.ex. Wireshark och Suricata är kompatibla med Napatechs nätverkskort. Napatechs nätverkskort stöder även programmeringsspråken C/C++ och Java med deras API (Sly Technologies, 2016). Nätverkskorten kommer även med ett webbläsarbaserat grafiskt användargränssnitt som man kan använda för att avläsa tidsstämplarna. (Napatech Pandion, 2016)

4.2.2 Microsemi MAX24288 nätverkskort

Microsemi MAX24288 tidsstämplaren skriver tidsstämplarna in i en FIFO-kö (eng. First In, First Out), varifrån mjukvaran kan läsa värdena. FIFO-kön har endast rum för åtta tidsstämplar. På grund av detta behöver inte processorn och nätverkskortet vara synkroniserade, eftersom tidsstämpeln skapas då ett datapaket har anlänt till nätverkskortet. Tidsstämpeln måste avläsas innan FIFO-kön har fyllts. Tidsstämplingen kan även konfigureras att ske då datapaketet anländer, då det tagits emot i sin helhet eller både och. Dessa tidsstämplar skapas med 1 nanosekunds resolution. Tidsstämplarna kan avläsas t.ex. med Wireshark. (Microsemi, 2016)

5 ALGORITMER BASERADE PÅ NOGGRANN TIDMÄTNING

Olika algoritmer kan kräva noggranna flygtidmätningar. En av dessa algoritmer är TTHCA-algoritmen.

5.1 TTHCA

I och med att efterfrågan på självkonfigurerande och infrastrukturlösa nätverk ökar blir mobila ad hoc nätverk (eng. Mobile Ad hoc Network, MANET) allt vanligare. Problemet med MANET är låg datasäkerhet med maskhålsattacker som en av de största riskerna. Maskhålsattackerna är svåra att undvika eftersom noderna maskerar sig som vanliga användarnoder i nätverket. Paketen som dirigerats via maskhålsnoden skickas vidare men deras säkerhet har äventyrats. (Karlsson, Dooley, & Pulkkis, 2011.)

Karlsson et al. (2011) har presenterat en ny algoritm för att upptäcka maskhålsattacker och försvara sig emot dem i MANET. Algoritmen baserar sig på analys av ett datapakets flygtid (eng. traversal time) och antalet hopp (eng. hop count) som datapaketet har gjort för att nå sin destination. Förkortningen som används för algoritmen är TTHCA (Traversal Time and Hop Count Analysis). Jämfört med tidigare lösningar förväntas denna algoritm konsekvent ge bättre resultat i form av högre maskhålsupptäckningsfrekvens och färre falskt positiva resultat.

För att förverkliga TTHCA bör man kunna uppskatta datapaketetens exakta flygtid mellan noderna. I denna avhandling kommer jag att undersöka hur noggrant flygtiden går att uppskatta. Syftet är att på basis av ett datapakets flygtid konsekvent, tillförlitligt och noggrant kunna uppskatta avståndet mellan två noder.

6 TIDMÄTNINGSEXPERIMENT

Den första tidmätningen som bör lyckas är processeringstidmätningen. För att kunna mäta noggranna flygtider och överföringstider bör processeringstidmätningen vara tillförlitlig och noggrann. Med hjälp av noggranna processeringstidmätningar kan man sedan uppnå noggranna flyg- och överföringstidmätningar, där man tar i beaktande processeringstiden.

6.1 Processeringstid

För att mäta processeringstiden har jag skapat ett C-program (Bilaga 3), som exekverar flera Inline assemblerinstruktioner. Före och efter dessa instruktioner avläses klockcykelregistrets innehåll, för att mäta hur många klockperioder instruktionernas exekvering består av. Dessutom har jag programmerat ett assemblerprogram för att mäta processeringstiden (Bilaga 7).

I moderna Intel-processorer har man tagit i beaktande att klockfrekvensen för en processor inte är konstant. På grund av detta har de moderna processorerna en flagga *constant_tsc*. Flaggan *constant_tsc* indikerar som inställd att klockcykelantalet sparat i TSC-registret baserar sig på processorkärnans maximala klockfrekvens. TSC-registrets innehåll representerar därför en tidmätning med en klockcykels noggrannhet.

Mätexperimenten är utförda på en dator med operativsystemet Linux och har körts dedikerat på en av datorns processorkärnor med klockfrekvensen 2,60 GHz. Processorn är flaggad för *constant_tsc*, vilket betyder alltså att en klockcykel blir 1/2,6 nanosekunder.

Experimenten bestod av 100 tidmätningar för vilka jag uträknade medeltalet, variansen och standardavvikelsen. Detta gjorde jag 20 gånger i *RDTSC*-experimentet och 30 gånger i *CLOCK_GETTIME*-experimentet. Därefter uträknade jag medeltalet, variansen och standardavvikelsen på medeltalen av de 20 respektive 30 mätresultaten för att uppnå det slutgiltiga resultatet.

6.1.1 Mätresultat

I Tabellerna 1-4 har jag dokumenterat mätresultaten för C-funktionen *CLOCK_GETTIME* och assembler-instruktionen *RDTSC*. I *CLOCK_GETTIME*-experimentet skapade jag två tidsstämplar med två funktionsanrop efter varandra och uträknade tidsskillnaden. I *RDTSC*-experimenten exekverade jag ett varierande antal *MOV*-instruktioner mellan *RDTSC*-tidsstämplarna med vilka jag uträknade tidsskillnaden. Jag utförde tre experiment med 1, 10 och 100 *MOV*-instruktioner mellan *RDTSC*-tidsstämplarna. Programkoden finns i Bilaga 3. Experimentet med 100 *MOV*-instruktioner illustreras grafiskt i Fig. 2.

Tabell 1. Mätresultat med CLOCK_GETTIME.

CLOCK_GETTIME (100 samples per #)			
#	AVG (ns)	VAR	STD
1	21	78	8
2	23	12	3
3	23	2	1
4	23	1	1
5	23	1	1
6	23	2	1
7	22	2	1
8	28	14	3
9	22	6	2
10	23	1	1
11	23	1	1
12	22	2	1
13	22	2	1
14	22	2	1
15	22	2	1
16	22	2	1
17	22	2	1
18	22	2	1
19	23	2	1
20	22	3	1
21	23	2	1
22	22	2	1
23	23	2	1
24	22	2	1
25	22	2	1
26	22	3	1
27	23	1	1
28	22	2	1
29	22	2	1
30	22	2	1
AVG	**22.53333**		
VAR	**1.315556**		
STD	**1.146977**		

Tabell 2. Mätresultat med RDTSC, 1 mov.

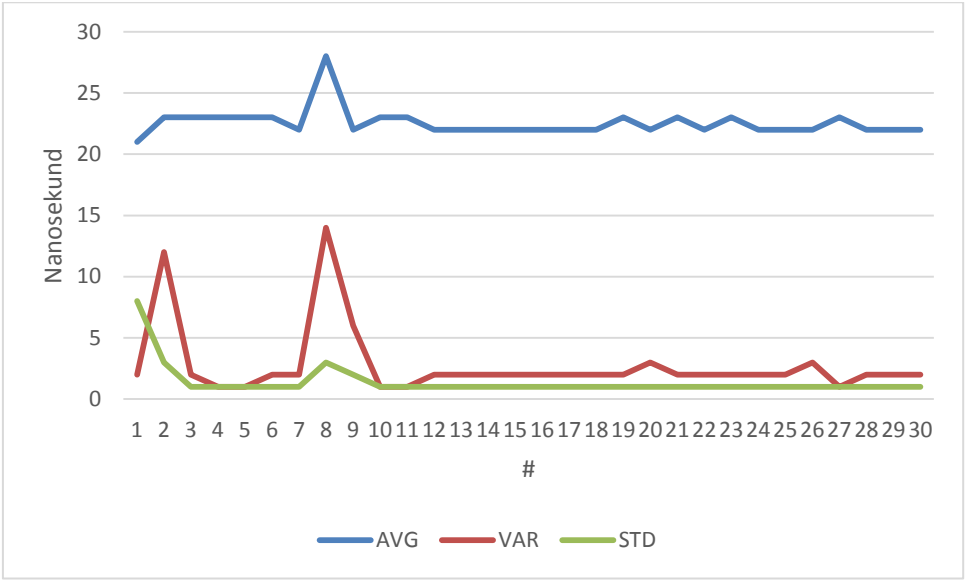
ASM RDTSC (1 MOV, 100 samples per #)				
#	AVG (cycles)	VAR	STD	AVG (ns)
1	25	14	3	10
2	26	15	3	10
3	25	14	3	10
4	26	14	3	10
5	26	14	3	10
6	24	13	3	9
7	24	14	3	9
8	25	12	3	10
9	24	14	3	9
10	26	16	4	10
11	26	15	3	10
12	25	14	3	10
13	24	14	3	9
14	25	13	3	10
15	24	14	3	9
16	25	13	3	10
17	24	13	3	9
18	24	12	3	9
19	25	13	3	10
20	24	13	3	9
			AVG	**9.56**
			VAR	**0.092826**
			STD	**0.304674**

Tabell 1. Mätresultat med RDTSC, 10 mov.

ASM RDTSC (10 MOV, 100 samples per #)				
#	AVG (cycles)	VAR	STD	AVG (ns)
1	26	16	4	10
2	26	16	4	10
3	29	18	4	11
4	26	15	3	10
5	26	17	4	10
6	26	15	3	10
7	29	15	3	11
8	26	15	3	10
9	29	16	4	11
10	29	15	3	11
11	29	14	3	11
12	26	15	3	10
13	29	14	3	11
14	26	14	3	10
15	29	16	4	11
16	30	14	3	12
17	26	14	3	10
18	26	15	3	10
19	26	15	3	10
20	26	15	3	10
			AVG	10.48
			VAR	0.35318
			STD	0.59429

Tabell 2. Mätresultat med RDTSC, 100 mov.

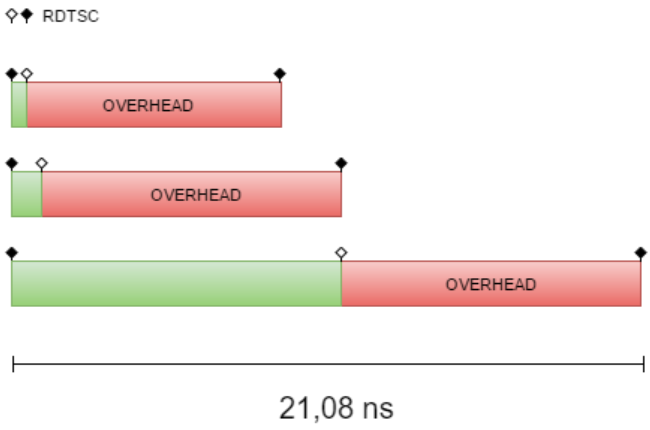
ASM RDTSC (100 MOV, 100 samples per #)				
#	AVG (cycles)	VAR	STD	AVG (ns)
1	55	14	3	21
2	46	24	4	18
3	56	26	5	22
4	55	15	3	21
5	56	37	6	22
6	55	22	4	21
7	55	12	3	21
8	55	11	3	21
9	55	13	3	21
10	56	13	3	22
11	55	10	3	21
12	55	12	3	21
13	56	13	3	22
14	55	13	3	21
15	55	12	3	21
16	55	11	3	21
17	55	12	3	21
18	55	13	3	21
19	56	13	3	22
20	55	13	3	21
			AVG	21.08
			VAR	0.630183
			STD	0.793841



Figur 2. Graf på mätresultatet med CLOCK_GETTIME.

Av dessa mätresultat kan man se att *CLOCK_GETTIME*-funktionen har en exekveringstid på ungefär 22,53 nanosekunder. Med noggrannare experimentering kunde man utreda vad den exakta exekveringstiden är. Ifall det exakta värdet kunde redas ut vore det möjligt att använda *CLOCK_GETTIME*-funktionen för överförings- och flygtidsmätningsexperimenten.

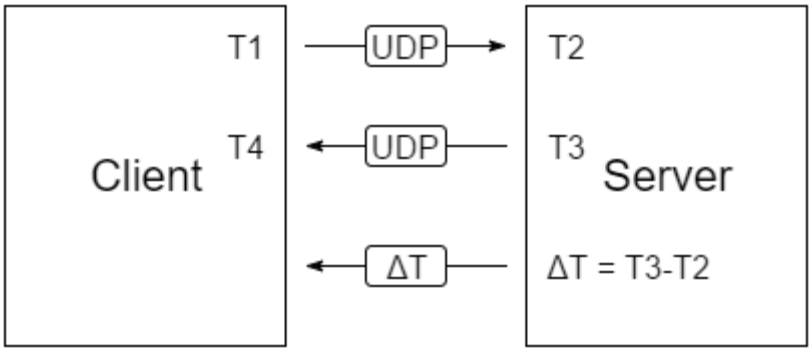
Med assembler-instruktionen *RDTSC* uppnåddes dock även noggrannare resultat med en exekveringstid på 9,56 nanosekunder med en *MOV* instruktion mellan tidsstämplarna. Exekveringstiden ökade till 10,48 nanosekunder med tio *MOV* instruktioner mellan tidsstämplarna. Detta skulle tyda på att själva exekveringstiden för *RDTSC* är ungefär 9,5 nanosekunder och att en *MOV* instruktion har exekveringstiden av 0,1 nanosekund. Då antalet *MOV* instruktioner ökar till hundra ökar exekveringstiden med 10,6 nanosekunder till ett medeltal av 21,08 nanosekunder. Om vi då räknar med att *RDTSC* har en exekveringstid på ungefär 9,5 nanosekunder så skulle det innebära att en *MOV* instruktion har exekveringstiden på ungefär 0,11 nanosekunder. Se Fig. 3. På grund av detta använder jag mig av *RDTSC* i fortsatta tidmätningsexperiment eftersom denna assemblerinstruktion har en kortare exekveringstid än *CLOCK_GETTIME*-funktionen.



Figur 3. Overhead-tiden.

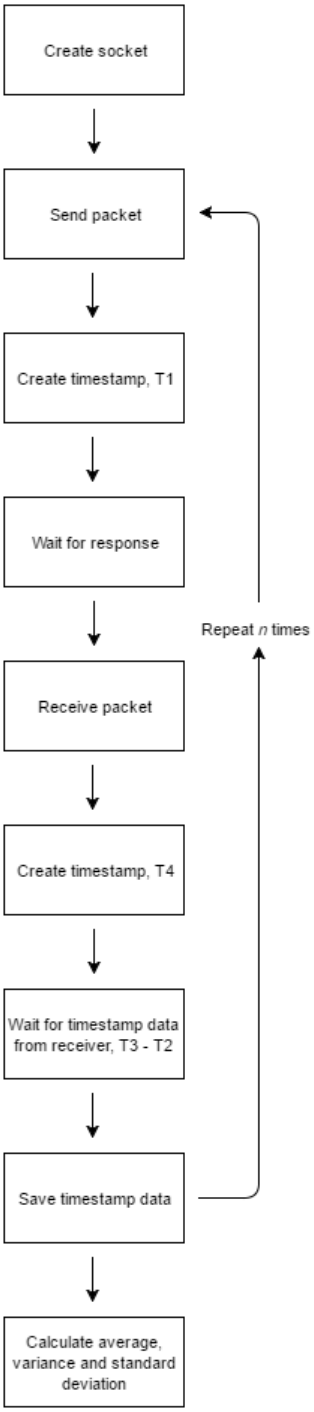
6.2 Överföringstid

I överföringstidmätningsexperimentet utreds hur noggrant man kan mäta ett datapakets överföringstid mellan två nätanslutna datorer. I experimentet används assembler-instruktionen RDTSC för att skapa fyra tidsstämplar. Dessa tidsstämplar används för att uträkna hur länge datapaketet har processerats för att sedan räkna ut överföringstiden enligt Fig. 4-6.

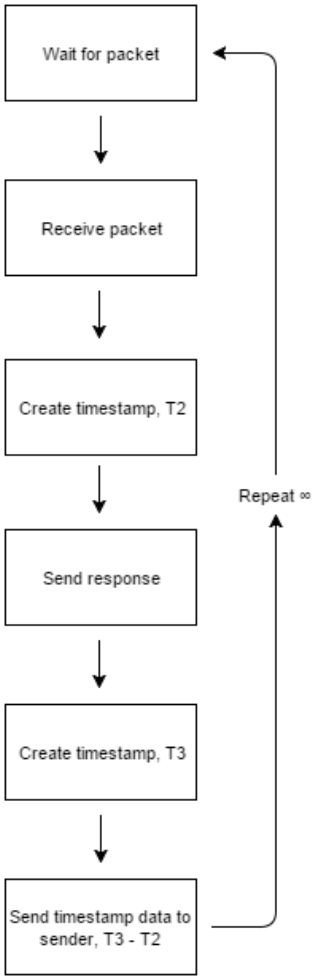


$$Överföringstid = \frac{T4 - T1 - \Delta T}{2}$$

Figur 4. Mätning av överföringstid.



Figur 5. Flödesschema för avsändaren.



Figur 6. Flödesschema för mottagaren.

6.2.1 Överföringstid i datanät

I mätexperimentet märktes tydligt att resultaten hade hög varians. Detta beror antagligen på varierande belastning i det datanät som datapaketen skickas igenom. Experimentet

utfördes i ett lokalnätverk. Mätresultaten är dokumenterade i Tabell 5. Programkoden för avsändaren och mottagaren finns i Bilaga 1 och 2.

På grund av den höga variansen i mätresultatet går det inte att uppskatta overhead-tiden. Detta innebär att överföringstiden inte heller går att uppskatta tillförlitligt. I överföringen mellan två datorer som är kopplade i samma nätverk finns det flera faktorer som skapar varians mellan mätgångerna. Dessa faktorer består av t.ex. nätverkets belastning, fördröjningen mellan nätverkskortet och processorn på datorerna samt varierande exekveringstider för programmen. I processeringstidmätningen var programmets exekvering kortare vilket inte gav utrymme för varians samt processorn kommunicerade inte med nätverkskortet.

Tabell 3. Mätresultatet för överföringstidsexperimentet.

TRANSFER TIME, RTT (100 samples per #)				
#	AVG (cycles)	VAR	STD	AVG (ms)
1	31537825	1.15E+19	33857045	17.52101389
2	33664136	1.77E+19	42126004	18.70229778
3	37415914	1.61E+20	40073552	20.78661889
4	28835818	5.14E+19	22670338	16.01989889
5	35248116	1.24E+20	35219166	19.58228667
6	41348211	2.00E+20	44731553	22.97122833
7	59049636	8.62E+21	293605316	32.80535333
8	33537304	1.71E+20	41375655	18.63183556
9	42772085	2.21E+20	46989775	23.76226944
10	53979209	2.44E+20	49391757	29.98844944
11	35415232	9.60E+19	30987859	19.67512889
12	24312250	3.43E+19	18517737	13.50680556
13	28626547	1.08E+20	32861694	15.90363722
14	32735944	1.55E+20	39346541	18.18663556
15	28789270	1.02E+20	31901322	15.99403889
16	33846793	1.22E+20	34936603	18.80377389
17	35625544	8.15E+19	28554587	19.79196889
18	36975655	1.32E+20	36382852	20.54203056
19	25918797	5.12E+19	22616408	14.39933167
20	35032710	3.59E+20	59945138	19.46261667
			AVG	2.858818806
			VAR	21.22167568
			STD	4.606699

6.2.2 Flygtid

För att utreda i praktiken hur noggrant man kan mäta flygtiden mellan två datorer planerades flygtidmätningsexperiment både inom- och utomhus. I experimenten är datorerna trådlöst anslutna till varandra, så att de kan direkt kommunicera med varandra. Då datorerna har direkt kontakt och fri sikt, borde datapaketens uppmätta flygtid vara det uppmätta avståndet mellan dem dividerat med ljusets hastighet. I inomhusexperimentet mäts avståndet mellan datorerna med måttband medan i GPS används i utomhusexperimentet.

Datapaketets flygtid uträknas från noggranna tidsstämplar. Tidsstämplarna skapas precis efter att ett datapaket tagits emot och precis efter att det skickats tillbaka. Med dessa tidsstämplar uträknas processeringstiden och flygtiden.

Detta experiment utförs med både C- och Assembler-programmen.

Mätresultat för inomhusexperimentet

I det första experimentet hade jag datorerna bredvid varandra och trådlöst uppkopplade till varandra. I detta experiment borde flygtiden i teorin ha ett värde lägre än en nanosekund p.g.a. att distansen mellan datorerna är under 0,3 meter. I mätresultatet som är dokumenterat i Tabell 6 ser man att den uppmätta flygtiden är mycket högre än den i teorin borde vara.

Den första kolumnen *Total, RTT* är antalet klockpulser som det tog att överföra paketet mellan datorerna. Kolumnen *Processing, PT* innehåller antalet klockpulser för processeringstiden hos mottagaren. Enligt formeln nedan har jag räknat ut flygtiden för mätresultaten i Tabell 6. I formeln använder jag den teoretiska maximala klockfrekvensen i båda processorerna, dvs. *C1* och *C2*.

$$Flygtid = \frac{RTT/C1 - PT/C2}{2}$$

Tabell 4. Mätresultat för C-programmet.

Time of Flight (Distance 0m, 100 samples per #)			
#	RTT	PT	Time of Flight, µs
1	10904077	46078	2.087855613
2	10131411	44224	1.939631137
3	8769550	44082	1.677761601
4	17413477	45573	3.339768728
5	14049489	54219	2.691138975
6	11018261	58971	2.107270175
7	10720516	47722	2.05223087
8	10675975	44682	2.044265095
9	8789037	44752	1.681376916
10	11095202	44864	2.124850133
11	11609711	45280	2.223712544
12	10442157	45016	1.999233984
13	10116782	45353	1.936595084
14	11722419	44277	2.245585168
15	9023995	43981	1.726713487
16	9846503	44119	1.884861602
17	11099546	43287	2.125996689
18	11288178	43985	2.162134515
19	10706023	43604	2.050256291
20	11570610	43579	2.216528722
		AVG	2.115888367
		VAR	0.127322495
		STD	0.356822778

Paketets flygtid mellan två datorer som är bredvid varandra är ungefär en nanosekund. Denna flygtid uppnås inte i mätexperimentet jag utfört. Mätresultatet är uppnått med C-programmet. Experimentet utfördes även med assembler-programmen och resulterade i liknande resolution men mycket högre varians. Det mätresultatet som uppnåddes med C-programmet har en noggrannhet av storleksklassen en mikrosekund. Samtidigt är variansen även med C-programmet för hög för att tillförlitligt kunna utesluta tidsstämplarnas overhead vilket gör det omöjligt att på basis av detta resultat räkna ut paketets flygtid.

Detta resultat beror antagligen på mycket samma faktorer som i överföringsexperimentet. Tiden det tar för nätverkskortet att förmedla informationen av ett inkommande datapaket till processorn samt exekveringstiden av programmen varierar. Dessa faktorer skulle inte vara relevanta ifall man använde sig av ett nätverkskort som skapade tidsstämpeln alltid då ett paket anlände eller skickades iväg. Då tidsstämplingen sker redan på nätverkskortet kan dessa faktorer alltså uteslutas vilket skulle resultera i ett noggrant och tillförlitligt mätresultat.

Utomhusexperimentet

Enligt mätresultatet som uppnåddes i det första flygtidsexperimentet inomhus framgick det att variansen redan då var för hög för att tillförlitligt kunna uppskatta flygtiden eller exekveringens overhead. På grund av detta utfördes inte några andra flygtidsexperiment.

7 SLUTSATSER OCH DISKUSSION

Programmen som användes för överförings- och flygtidsexperimenten kunde möjligtvis optimeras och finslipas, men på grund av det mätresultat som uppnåddes är det osannolikt att den målsatta resolutionen skulle nås. Programmet som användes för processeringstidmätningen behöver inte optimeras eftersom det utsatta målet nåddes då variansen var tillräckligt låg för att utesluta exekveringens overhead.

KÄLLOR

BinaryTides, 2012. Programming raw udp sockets in C on Linux
Tillgänglig:
http://www.binarytides.com/raw-udp-sockets-c-linux/
Hämtad 4.4.2016

Cruse, A. 2009. CS 210: Assembly Language and Systems Programming (Spring 2009), [www].
Tillgänglig: http://www.cs.usfca.edu/~cruse/cs210s09/
Hämtad 2.7.2016

Die.net, clock_gettime(3) – Linux man page, [www].
Tillgänglig: http://linux.die.net/man/3/clock_gettime
Hämtad 4.3.2016

Gal, T., & Morgan, C. 2014. SharpPcap – A Packet Capture Framework for .NET, [www].
Tillgänglig: http://www.codeproject.com/Articles/12458/SharpPcap-A-Packet-Capture-Framework-for-NET
Hämtad 4.3.2016

Karlsson, J., Laurence, D., & Pulkkis, G. 2011. A New MANET Wormhole Detection Algorithm Based on Traversal Time and Hop Count Analysis, [pdf].
Tillgänglig: http://www.mdpi.com/1424-8220/11/12/11122/htm
Hämtad 5.2.2015

Microsemi, 2016. MAX24288 IEEE 1588 Packet Timestamper and Clock and 1 Gbps Parallel-to-Serial MII Converter, s. 47-48, [pdf].
Tillgänglig:
http://www.microsemi.com/index.php?option=com_docman&task=doc_download&gid=126640

Hämtad 20.6.2015

Mono, [www].
Tillgänglig: http://www.mono-project.com/
Hämtad 20.6.2016

Napatech. Hardware Time Stamp, [www].
Tillgänglig: http://w3new.napatech.com/features/time-precision/hardware-time-stamp
Hämtad 9.5.2015

Napatech Pandion. Network recorder, [pdf].
Tillgänglig: http://marketing.napatech.com/acton/attachment/14951/f-005b/1/-/-/-/-/Pandion%20Network%20Recorder_%20Napatech%20Data%20Sheet.pdf
Hämtad 4.3.2016

Sly Technologies, Capture and Transmit Cards by Napatech, [www].
Tillgänglig: http://slytechs.com/products/napatech
Hämtad 4.3.2016

Wibowo, S. B., Klepal, M., & Pesch, D. 2009. Time of Flight Ranging using Off-the-shelf IEEE802.11 WiFi Tags
Tillgänglig:
www.locon-eu.com/Documents/POCA%20Conference_CIT_Sigit.pdf
Hämtad 4.3.2016

BILAGOR

Bilaga 1, Flygtidmätning – Avsändare, C

```c
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include <time.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<math.h>

#define SERVER "10.42.0.1"
#define BUFLEN 512
#define PORT 8888

void die(char *s)
{
    perror(s);
    exit(1);
}

int main(void)
{
    struct sockaddr_in si_other;
    uint32_t t1 = 0, t2 = 0;
    int s, i, j, k, n = 100, x, slen=sizeof(si_other);
    long long slo = 0, elo = 0, TSS[n], TSR[n], TOTS = 0, TOTR = 0, AVGS
= 0, AVGR = 0, VARS = 0, VARR = 0, STDS = 0, STDR = 0, TMP, ts1, ts2;
    char buf[BUFLEN], message[BUFLEN];
    FILE *output;
    system("lscpu | grep 'MHz'");

    //Create message to be sent
    sprintf(message, "%s", "String that is sent");

    //Create socket
    if ( (s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
    {
         die("socket");
    }
        memset((char *) &si_other, 0, sizeof(si_other));
        si_other.sin_family = AF_INET;
        si_other.sin_port = htons(PORT);

        if (inet_aton(SERVER , &si_other.sin_addr) == 0)
        {
          fprintf(stderr, "inet_aton() failed\n");
          exit(1);
        }

    //Start sending 20*n times
    for(k=0; k<20; k++){
        for (i=0; i<n; i++)
        {
                //Send packet
```

```
                if (sendto(s, message, strlen(message) , 0 , (struct
sockaddr *) &si_other, slen)==-1)
                {
                    die("sendto()");
                }
                memset(buf,'\0', BUFLEN);
                //Create timestamp t1
                asm (
                "rdtsc;"
                "movl %%eax, %0;"
                :"=r" (t1)
                );

                //Wait for response & receive packet
                if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *)
&si_other, &slen) == -1)
                {
                        die("recvfrom()");
                }
                //Create timestamp t2
                asm (
                "rdtsc;"
                "movl %%eax, %0;"
                :"=r" (t2)
                );

                //Wait for timestamp data from receiver, t4 - t3
                if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *)
&si_other, &slen) == -1)
                {
                        die("recvfrom()");
                }

                //Save timestamp data
                ts1 = t1;
                ts2 = t2;
                if(ts1 < ts2){
                    TSS[i] = ts2-ts1;
                }
                TSR[i] = atoi(buf);
                TOTS = TOTS + TSS[i];
                TOTR = TOTR + TSR[i];
                printf("%lld    %lld\n", TSS[i], TSR[i]);
                sleep(1);
        }

        //Process, save and print average, variance and standard
deviation data

        //Calculate sender values
        AVGS = TOTS/n;
        TMP = 0;
        for(j=0; j<n; j++)
        {
            TMP += (AVGS-TSS[j])*(AVGS-TSS[j]);
        }
        VARS = TMP/n;
        STDS = sqrt(VARS);

        //Calculate receiver values
        AVGR = TOTR/n;
        TMP = 0;
        for(j=0; j<n; j++)
        {
            TMP += (AVGR-TSR[j])*(AVGR-TSR[j]);
        }
        VARR = TMP/n;
```

```
            STDR = sqrt(VARR);

            output = fopen("tsdata", "a");

            printf("#%d:  s=%lld    r=%lld\n", k+1, AVGS,AVGR);

            fprintf(output,"%d  %lld    %lld\n", k+1, AVGS, AVGR);
            fclose(output);
            TOTS = 0;
            AVGS = 0;
            VARS = 0;
            STDS = 0;
            TOTR = 0;
            AVGR = 0;
            VARR = 0;
            STDR = 0;
        }
        system("lscpu | grep 'MHz'");
    }
```

Bilaga 2, Flygtidmätning – Mottagare, C

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define BUFLEN 512  //Max length of buffer
#define PORT 8888   //The port on which to listen for incoming data

void die(char *s)
{
    perror(s);
    exit(1);
}

void startServer(){
    struct sockaddr_in si_me, si_other;

        int s, i, slen = sizeof(si_other) , recv_len;
    unsigned long long t3, t4;
    unsigned long flags;
    char buf[BUFLEN];

    system("lscpu | grep 'MHz'");

     //create a UDP socket
     if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
     {
         die("socket");
     }

        // zero out the structure
        memset((char *) &si_me, 0, sizeof(si_me));

        si_me.sin_family = AF_INET;
        si_me.sin_port = htons(PORT);
        si_me.sin_addr.s_addr = htonl(INADDR_ANY);

        //bind socket to port
        if( bind(s , (struct sockaddr*)&si_me, sizeof(si_me) ) == -1)
     {
            die("bind");
        }

        //Wait for packet
        while(1)
        {
           fflush(stdout);

        //try to receive some data, this is a blocking call
        if ((recv_len = recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *)
&si_other, &slen)) == -1)
        {
            die("recvfrom()");
        }
    //Create timestamp, t3
    __asm__ __volatile__ ("rdtsc" : "=A" (t3) );

    sprintf(buf, "%s", "Data");

    //Send response
```

```
        if (sendto(s, buf, 9, 0, (struct sockaddr*) &si_other, slen) ==
-1)
        {
            die("sendto()");
    }

    //Create timestamp, t4
        __asm__ __volatile__ ("rdtsc" : "=A" (t4) );

    //Send timestamp data to sender, t4-t3
    sprintf(buf, "%lld", t4-t3);
    if (sendto(s, buf, 9, 0, (struct sockaddr*) &si_other, slen) == -1)
        {
            die("sendto()");
        }
    printf("Sent ts: %lld\n", t4-t3);
    //restore_flags(flags);
    }

//__asm__("sti");

    close(s);
}

int main()
{
    startServer();
        return 0;
}
```

Bilaga 3, Processeringstidmätning, C

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<stdint.h>
#include <time.h>
#include<math.h>

int main(void)
{

    uint32_t shi32 = 0, slo32 = 0, ehi32 = 0, elo32 = 0;
    short a = 0;
    int i = 0, j = 0, k = 0, n = 100, x;
    long long shi = 0, slo = 0, ehi = 0, elo = 0, TOTA = 0, AVGA = 0,
VARA = 0, STDA = 0, TSA[n], TOTC = 0, AVGC = 0, VARC = 0, STDC = 0,
TSC[n], temp = 0;
    struct timespec t1, t2, res;
    int id = CLOCK_REALTIME;
    FILE *output;
    system("lscpu | grep 'MHz'");
    for(k=0; k<20; k++){


        for (i=0; i<n; i++){

            /*
            clock_getres(id, &res);
            clock_gettime(id, &t1);
            clock_gettime(id, &t2);

            TSC[i] = t2.tv_nsec-t1.tv_nsec;
            TOTC = TOTC + TSC[i];
            */

            asm (
            "rdtsc;"
            "movl %%eax, %%ecx;"

            "rdtsc;"
            "movl %%ecx, %0;"
            "movl %%eax, %1;"

            :"=r" (slo32),
             "=r" (elo32)
            );


            //shi = shi32;
            slo = slo32;
            //ehi = ehi32;
            elo = elo32;

            TSA[i] = elo-slo;
            TOTA = TOTA + TSA[i];

        }
        /*
        //Calculate and print AVERAGE, VARIANCE and STANDARD DEVIATION
for CLOCK_GETTIME
        AVGC = TOTC/n;
        temp = 0;
        for(j=0; j<n; j++){
            temp += (AVGC-TSC[j])*(AVGC-TSC[j]);
        }
```

```
        VARC = temp/n;
        STDC = sqrt(VARC);

        output = fopen("ptdata", "a");

        printf("#%d:  %lld\n", k+1, AVGC);
        system("lscpu | grep 'MHz'");

        fprintf(output,"%d  %lld    %lld    %lld\n", k+1, AVGC, VARC,
STDC);
        fclose(output);
        TOTC = 0;
        AVGC = 0;
        VARC = 0;
        STDC = 0;
        */
        //Calculate and print AVERAGE, VARIANCE and STANDARD DEVIATION
for ASM RDSTC
        AVGA = TOTA/n;
        temp = 0;
        for(j=0; j<n; j++){
            temp += (AVGA-TSA[j])*(AVGA-TSA[j]);
        }
        VARA = temp/n;
        STDA = sqrt(VARA);

        output = fopen("ptdata", "a");

        printf("#%d:  %lld\n", k+1, AVGA);

        fprintf(output,"%d  %lld    %lld    %lld\n", k+1, AVGA, VARA,
STDA);
        fclose(output);
        TOTA = 0;
        AVGA = 0;
        VARA = 0;
        STDA = 0;
        //sleep(1);
    }
system("lscpu | grep 'MHz'");
}
```

Bilaga 4, Aktiveringsprogram, C

```
int main() {
while(1);
}
```

Bilaga 5, Flygtidmätning, Avsändare, Assembler

```
# manifest constants
      .equ  sys_socketcall, 102
      .equ  SYS_SOCKET, 1           # from <linux/net.h>
      .equ  SYS_BIND, 2
      .equ  SYS_SENDTO, 11
      .equ  SYS_RECVFROM, 12
      .equ  SYS_RECV, 10
      .equ  STDOUT, 1         # from <unistd.h>
      .equ  PF_INET, 2        # from <bits/socket.h>
      .equ  SOCK_DGRAM, 2           # from <bits/socket.h>
      .equ  IPPROTO_UDP, 17         # from <linux/in.h>
      .equ  dev_STDOUT, 1
      .equ  sys_WRITE, 1
      .equ  sys_EXIT, 60



      .section    .data
# the message-text to be transmitted
data: .asciz      "d\n"
dlen: .int  . - data          # length of datagram's data

# our buffer for the server's reply
buf:  .space      1500              # buffer for received packet
len:  .int  . - buf                 # size of the receive buffer

# the array for the socketcall parameters
args: .int  PF_INET, SOCK_DGRAM, IPPROTO_UDP, 0, 0, 0, 0, 0

# the 'sock_address' data-structure
addr: .short      PF_INET                 # ID for the protocol-family
port: .short      60533                   # the server's port-number
      .byte 192, 168, 0, 14         # the server's IP-address
      .space      8                 # padding for structure-size
alen: .int  16                # variable needed for length

# miscellaneous data
sock: .int  -1                # place to put socket's ID

tscval:     .quad 0
temp: .quad 0
tscmsg:     .ascii      "\n TimeStamp Counter: "
tscbuf:     .ascii      "                   \n\n"
tsclen:     .quad . - tscmsg


      .section    .text
_start:
      # create the socket using preinitialized arguments
      mov   $sys_socketcall, %eax   # system-call ID-number
      mov   $SYS_SOCKET, %ebx # subfunction ID-number
      lea   args, %ecx        # pointer to arguments
      int   $0x80             # invoke kernel service
      mov   %eax, sock        # save socket's handle

      # prepare argument-array for our 'sendto' system-call
```

```
        mov   sock, %eax
        mov   %eax, args+0            # sock
        lea   data, %eax
        mov   %eax, args+4            # &data
        mov   dlen, %eax
        mov   %eax, args+8            # dlen
        movl  $0, args+12       # flags
        movl  $addr, args+16          # &addr
        mov   alen, %eax
        mov   %eax, args+20           # alen
        rolw  $8, port          # swap bytes (for network-order)

        # invoke system-sall to transmit our datagram
        mov   $sys_socketcall, %eax   # system-call ID-number
        mov   $SYS_SENDTO, %ebx # subfunction ID-number
        lea   args, %ecx        # pointer to arguments
        int   $0x80             # invoke kernel service

        # read the TimeStamp Counter register
        rdtsc
        mov   %eax, temp

        # now marshall the arguments needed for 'recvfrom'
        mov   sock, %eax
        mov   %eax, args+0            # sock
        movl  $buf, args+4            # &buf
        mov   len, %eax
        mov   %eax, args+8            # len
        movl  $0, args+12       # flags

        # invoke system-call to receive server's reply
        mov   $sys_socketcall, %eax   # system-call ID-number
        mov   $SYS_RECV, %ebx         # subfunction ID-number
        lea   args, %ecx        # pointer to arguments
        int   $0x80             # invoke kernel service
        mov   %eax, len         # save message's length

        rdtsc
        sub   temp, %eax
        mov   %eax, tscval

        # format the register-value for output
        mov   tscval, %rax
        lea   tscbuf, %rdi
        mov   $10, %rbx
        xor   %rcx, %rcx

nxdiv:
        xor   %rdx, %rdx
        div   %rbx
        push  %rdx
        inc   %rcx
        or    %rax, %rax
        jnz   nxdiv
nxdgt:
        pop   %rdx
        add   $'0', %dl
        mov   %dl, (%rdi)
        inc   %rdi
        loop  nxdgt
```

```
        # write the report-message
        mov    $sys_WRITE, %rax
        mov    $dev_STDOUT, %rdi
        lea    tscmsg, %rsi
        mov    tsclen, %rdx
        syscall

        #loop _start
end:
        # terminate this program
        mov    $sys_EXIT, %rax
        xor    %rdi, %rdi
        syscall

        .global     _start
        .end
```

Bilaga 6, Flygtidmätning, Mottagare, Assembler

```
        # manifest constants
        .equ  sys_WRITE, 1
        .equ  sys_SOCKET, 41
        .equ  sys_SENDTO, 44
        .equ  sys_RECVFROM, 45
        .equ  sys_BIND, 49
        .equ  sys_GETSOCKNAME, 51
        .equ  sys_EXIT, 60
        .equ  STDOUT, 1
        .equ  AF_INET, 2
        .equ  SOCK_DGRAM, 2
        .equ  IPPROTO_UDP, 17
        .equ  BUFSIZ, 1500
        .equ  dev_STDOUT, 1


        .section    .data

# internet socket-address structure for this host
haddr:      .short      AF_INET           # for sin_family
        .short      0           # for sin_port
        .byte 0, 0, 0, 0  # for sin_addr
        .space      8           # for padding
halen:      .quad . - haddr   # structure-length

# internet socket-address structure for the client
caddr:      .short      AF_INET           # for sin_family
        .short      0           # for sin_port
        .byte 0, 0, 0, 0  # for sin_addr
        .space      8           # for padding
calen:      .quad . - haddr   # structure-length

sock: .quad -1          # place to keep socket-handle

pmsg: .ascii      "Socket has port-number "
pnum: .ascii      "                    \n"
plen: .quad . - pmsg

buf:  .space      BUFSIZ            # buffer for received message
len:  .quad BUFSIZ            # length of the buffer space

tscval:     .quad 0
temp: .quad 0
tscmsg:     .ascii      "\n TimeStamp Counter: "
tscbuf:     .ascii      "                  \n\n"
tsclen: .quad     . - tscmsg


        .section    .text
_start:
        # create an internet datagram socket
        mov   $sys_SOCKET, %rax # system-call ID-number
        mov   $AF_INET, %rdi          # domain
```

```
mov   $SOCK_DGRAM, %rsi # type
mov   $IPPROTO_UDP, %rdx      # protocol
syscall                       # invoke kernel service
mov   %rax, sock        # save socket's handle

# bind our socket to a generic sock-address structure
mov   $sys_BIND, %rax         # system-call ID-number
mov   sock, %rdi        # socket-handle
lea   haddr, %rsi       # pointer to sock-address
mov   halen, %rdx       # length of sock-address
syscall                       # invoke kernel service

# find the system-assigned port-number and display it
mov   $sys_GETSOCKNAME, %rax  # system-call ID-number
mov   sock, %rdi        # socket-handle
lea   haddr, %rsi       # pointer to sock-address
lea   halen, %rdx       # pointer to sock-addr length
syscall                       # invoke kernel service

movzxw      haddr+2, %rax           # get assigned port-number
rol   $8, %ax                 # convert to host byte-order
lea   pnum, %rdi        # point to number buffer
call  num2asc                 # convert port to ascoo

mov   $sys_WRITE, %rax  # system-call ID-number
mov   $STDOUT, %rdi           # device-file ID-number
lea   pmsg, %rsi        # point to message string
mov   plen, %rdx        # length of message-string
syscall                       # invoke kernel service

# read any datagram sent to our socket's port-number
mov   $sys_RECVFROM, %rax     # system-call ID-number
mov   sock, %rdi        # socket's ID-number
lea   buf, %rsi         # buffer's address
mov   $BUFSIZ, %rdx           # buffer's length
xor   %rbx, %rbx        # flags (none set)
lea   caddr, %r8        # sock-address pointer
lea   calen, %r9        # sock-address length-pointer
syscall                       # invoke kernel service
mov   %rax, len         # save received message length

# read timestamp counter
rdtsc
mov   %eax, temp

# display the received message
#mov  $sys_WRITE, %rax  # system-call ID-number
#mov  $STDOUT, %rdi           # device-file ID-number
#lea  buf, %rsi         # message's address
#mov  len, %rdx         # message's length
#syscall                # request kernel service

# echo the received message back to the client
mov   $sys_SENDTO, %rax # system-call ID-number
mov   sock, %rdi        # socket's ID-number
lea   buf, %rsi         # message's address
mov   len, %rdx         # message's length
xor   %rbx, %rbx        # flags (none set)
lea   caddr, %r8        # sock-address pointer
mov   calen, %r9        # sock-address length
```

```
        syscall                         # request kernel service

        # read timestamp counter
        rdtsc
        sub     temp, %eax
        mov     %eax, tscval

        # format the register-value for output
        mov     tscval, %rax
        lea     tscbuf, %rdi
        mov     $10, %rbx
        xor     %rcx, %rcx

        # write the report-message
        mov     $sys_WRITE, %rax
        mov     $dev_STDOUT, %rdi
        lea     tscmsg, %rsi
        mov     tsclen, %rdx
        syscall

exit:
        # terminate this program
        mov     $sys_EXIT, %rax         # system-call ID-number
        xor     %rdi, %rdi          # use zero as exit-code
        syscall                         # invoke kernel service

num2asc: # converts the value in RAX to a decimal string at (RDI)

        push    %rax                # save caller's registers
        push    %rbx
        push    %rcx
        push    %rdx
        push    %rdi

        mov     $10, %rbx           # decimal-system's radix
        xor     %rcx, %rcx          # initialize digit count

nxdiv:      xor     %rdx, %rdx          # prepare 128-bit dividend
        div     %rbx                # divide by system radix
        push    %rdx                # push the remainder
        inc     %rcx                # and count this remainder
        or      %rax, %rax          # was quotient zero?
        jnz     nxdiv               # no, do another division
nxdgt:      pop     %rdx                # pop previous remainder
        add     $'0', %dl           # convert int to numeral
        mov     %dl, (%rdi)         # write numeral to buffer
        inc     %rdi                # advance buffer-index
        loop    nxdgt               # again for other remainders


        pop     %rdi                # recover saved registers
        pop     %rdx
        pop     %rcx
        pop     %rbx
        pop     %rax
        ret                         # return control to caller


        .global     _start                  # make entry-point public
        .end                        # no more to be assembled
```

Bilaga 7, Processeringtidmätning, Assembler

```
        .equ  dev_STDOUT, 1
        .equ  sys_WRITE, 1
        .equ  sys_EXIT, 60


        .section    .data
tscval:     .quad 0
temp: .quad 0
tscmsg:     .ascii      "\n TimeStamp Counter: "
tscbuf:     .ascii      "                    \n\n"
tsclen:     .quad . - tscmsg


        .section    .text
_start:
        # read the TimeStamp Counter register
        rdtsc
        mov   %eax, temp
        rdtsc
        sub   temp, %eax
        mov   %eax, tscval

        # format the register-value for output
        mov   tscval, %rax
        lea   tscbuf, %rdi
        mov   $10, %rbx
        xor   %rcx, %rcx
nxdiv:
        xor   %rdx, %rdx
        div   %rbx
        push  %rdx
        inc   %rcx
        or    %rax, %rax
        jnz   nxdiv
nxdgt:
        pop   %rdx
        add   $'0', %dl
        mov   %dl, (%rdi)
        inc   %rdi
        loop  nxdgt

        # write the report-message
        mov   $sys_WRITE, %rax
        mov   $dev_STDOUT, %rdi
        lea   tscmsg, %rsi
        mov   tsclen, %rdx
        syscall

        loop _start

        # terminate this program
        mov   $sys_EXIT, %rax
        xor   %rdi, %rdi
        syscall

        .global     _start
        .end
```