Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

NAGYPONTOSSÁGÚ ÓRASZINKRONIZÁCIÓ IEEE 1588 ALAPON BEÁGYAZOTT LINUX RENDSZEREKBEN

TDK dolgozat

Ferencz Bálint

VI. évf, villamosmérnök szakos hallgató

Konzulens:

Dr. Kovácsházy Tamás docens – khazy@mit.bme.hu
Méréstechnika és Információs Rendszerek Tanszék

Tartalomjegyzék

Összefoglaló

Napjainkban sokféle architektúra igyekszik megoldást találni a szinkronizált feladatvégzésre az elosztott rendszerekben. A már meglevő kommunikációs csatornákat felhasználó óraszinkronizációs algoritmusok egy olcsó megoldást nyújtanak a globális óra terjesztésére. Az elosztott rendszer kifejezés sokféle méretű és bonyolultságú rendszert takar, a legkisebb beágyazott rendszerektől a nagyteljesítményű klaszterekig. A legtöbb rendszer által használt hálózat multicast üzenetszórásra képes, ezáltal a különböző óraszinkronizációs eljárások terjednek ennek kiaknázására.

A távoli események pontos keletkezési idejének ismerete szükséges ahhoz, hogy a adat/jelfeldolgozó eszközök a rendszerben történő mérések, tranzakciók megkülönböztethessék, és megfelelő módon sorba rendezhessék. Példaképp a nagy tőzsdéket kiszolgáló számítógépes infrastruktúrákban nem ritka a másodpercenként egymillió tranzakció sem, ami azt jelenti, hogy azok keletkezési idejét legalább mikroszekundum pontossággal ismernünk kell. A villamosenergia-elosztó hálózat alállomásaiban történő tranziensmérések szintén nagy pontosságú időmérést, és kifinomult jelfeldolgozási algoritmusokat igényelnek a hiba helyének pontos meghatározására. Szórakoztatóelektronika területén is léteznek iparági törekvések a kommunikációs csatornák egységesítésére (802.11 AVB), azért, hogy a berendezések fizikai felépítése egyszerűsödjön, és jobban integrálódjanak az otthon ambiens rendszerei közé. A szabvány által definiált alszabványok közt van egy IEEE 1588 alapú óraszinkronizációs protokoll (802.11AS), mely segítségével az audio-video jelfolyam kis jitterrel rekonstruálható a végpontokon.

A dokumentum egy IEEE 1588 alapú megoldást mutat be nem valósidejű Linux környezetben, demonstrálva annak alkalmazhatóságát nagy pontosságú (< 1 µs), helyi hálózaton működő óraszinkronizációs rendszerekben. Bemutatásra kerül egy módszer, hogy miként integráljunk nagy pontosságú frekvenciaforrást a rendszerbe a felhasznált hálózati csatoló hardveres szolgáltatásait felhasználva. A felépített rendszer pontosságát külső független referenciaforrás ellenében vizsgáltam meg. A bemutatott megoldás jól hordozható beágyazott rendszerekbe is, különösebb architekturális változtatások nélkül. A dolgozat végül összefoglalja a különségeket a valósidejű és nem valósidejű rendszerekben megkövetelt tulajdonságok közt a nagypontosságú időszinkronizáció tekintetében.

Abstract

Nowadays several architectures pursuit the goal of synchronized operation in the field of distributed systems. The usage of the shared communication media to execute the clock synchronization algorithms provides a cost-effective solution to maintain the common global clock of the nodes. The distributed system term covers a wide variety of system designs ranging from the smallest embedded systems to high performance clusters. Most of these systems use multicast capable communication medium to communicate with each other (notably Ethernet), therefore a number of clock synchronization algorithms were built on this architecture.

In order to distinguish measurements, transactions the precise creation date of these remote events should be known to reestablish the correct event order in the data processing node. For example in large-scale stock market trading the one million transaction per second rate is common, thus we need to know the timestamps of these transactions at least one microsecond accuracy. In the measurement of the transients of a power substation also requires delicate measurement algorithms and correct clocks of the measurement devices to extract the actual cause of the malfunction. Industry efforts are also made to equip the modern home entertainment systems with a single communication standard (802.11 AVB) in order to simplify the physical structure of the devices, and also increase their integration in the ambient home conception. Among the necessary sub-standards a simplified IEEE 1588 based clock synchronization protocol is defined (802.11AS) to deliver the audio-video stream in a low jitter manner.

In this paper an IEEE 1588 based solution is introduced in non-real-time Linux environment to demonstrate the high accuracy (< 1 μs) clock synchronization in local area networks. We present a solution to integrate an absolute frequency reference into the communication hardware by using the hardware capabilities of the Ethernet adapters. The accuracy of the system was verified against external reference clock by also using the hardware assistance of the Ethernet network interfaces. The presented method scales well into embedded systems, all of the functionality can be maintained without architectural changes. The paper summarizes the differences between the RT and non-RT systems in respect of the required capabilities in order to attain the desired accuracy.

Bevezetés

Napjaink üzleti, telekommunikációs, mérésadatgyűjtő és szórakoztatóelektronikai rendszereit tervező mérnökök az elosztott felépítésnek köszönhetően egyre gyakrabban szembesülnek az elosztott rendszer integritását érintő kérdésekkel. Ezek közül az egyik legfontosabb a globális idő kezelésének problémaköre.

A korábbi TDK dolgozatomban demonstráltam, hogy miként lehetséges egy Linux alapú rendszerek szoftveres rendszeróráit szinkronizálni nagy pontossággal az IEEE 1588 szabványt megvalósító szinkronizációs megoldások segítségével. Megoldási javaslatot tettem a pontosság további növelésére, és a nagypontosságú ellenőrzés módszereire. Ezen dolgozat körüljárja az azóta elvégzett munkát, bemutatja a pontosság növelésének egy működő példáját, valamint egy megoldást a rendszer validációjára, és a rendszer fizikai környezetébe történő beillesztésére. A dolgozat a terminológiák, és az időszinkronizáció történeti ismertetése után röviden vázolja a PTP algoritmus működését, majd bemutatja a tervezési megfontolásokat. Ezek után a jelenleg használt mérési elrendezés leírása következik, ahol kitérek a komponensek megválasztásának részletes okaira, és ismertetem az implementáció működését. A mérési eredmények ismertetése és kiértékelése után, bemutatom a rendszer skálázódását kisebb beágyazott környezetbe, végül javaslatokat teszek a továbbfejlesztési irányokra.

A dolgozatban használt terminológia

A dolgozat további részeinek könnyű követhetősége végett röviden összefoglalom a dolgozatban használt gyakoribb rövidítéseket, kifejezéseket, hogy az olvasó számára mindenütt egyértelmű legyen a dolgozatban dokumentált rendszer működése. Mivel a dolgozat magyar nyelven íródott, így igyekeztem a magyar szaknyelvben meghonosodott kifejezéseket használni. Ettől csak akkor tekintettem el, ha az adott kifejezésnek nincsen széles körben elterjedt magyar nyelvű egyértelmű megfelelője.

PTP – Precision Time Protocol, *IEEE 1588*-as szabványban foglalt ajánlott rövidítése az algoritmusnak

óra *stabilitása* – angolul: *stability,* az óra elvárt konstans frekvenciához viszonyított hibája

óra *pontossága* – angolul: *accuracy,* az óra által mutatott aktuális idő eltérése a referenciától

óra *felbontása* – angolul: *precision,* a vizsgált óra időkvantuma

ofszet – angolul: *offset,* két óra által mutatott idő különbsége

frekvenciahiba – angolul: *skew*, órák frekvenciája közti különbség

végpont – angolul: *node,* a logikai portokat befogadó végeszközök

időalapok – angolul: *synchronization master*

szinkronizálódó fél – angolul: *synchronization slave*

késleltetés – angolul: *delay*

késleltetés ingadozás – angolul: *jitter*

ppm – angolul: *particle-per-million*, egy mennyiség *egymilliomod része*

ppb – angolul: *particle-per-billion*, egy mennyiség *egymilliárdod része*

kötés – angolul: *binding*, alrendszerek közti interfészek

rendszermag – angolul: *kernel*

felhasználói programok – angolul: *userspace*

PPS – pulse per second, önmagában az 1 Hz-es négyszögjelet jelöl, valamint a Linux egy szinkronizációs alrendszerét, *nPPS* alatt n-szeres frekvenciájú jelről beszélünk

GPS – Global Positioning System, az USA által üzemeltetett helymeghatározási és óraszinkronizációs rendszer

TAI – International Atomic Time, az elemi nemzetközi időskála, amely egyezményesen definiálja a pontos időt a Földön, tisztán a céziumatom fizikai tulajdonságait figyelembevéve

UTC – Coordinated Universal Time, a TAI-n alapuló időskála, amely figyelembe vesz bizonyos Föld forgásából adódó jelenségeket, a TAI-hez szökőmásodpercek hozzáadásával

MAC – angolul: Medium Access Controller, a hálózati hozzáférést adminisztráló funkcionális blokk, mely még nincs közvetlen fizikai kapcsolatban az átvitelt bonyolító csatornával

PHY – a hálózati eszközök fizikai réteghez illesztő (fizikai réteg vezérlő) áramkörének általános elnevezése

L2 – ezzel a rövidítéssel hivatkozom a nyers Ethernet keretekben továbbított időszinkronizációs üzenetek továbbítási módjára

L4 – ezzel a rövidítéssel pedig az UDP/IP hordozóba ágyazott üzenetek továbbítási módját jelölöm

Az időszinkronizáció története az informatikában

Az órajel szinkronizáció igénye a legelső hardveres interfészek megjelenésével együtt jelent meg, hiszen a kommunikációhoz elengedhetetlen, hogy az adó, valamint a vevő azonos ütemre legyenek képesek adatokat cserélni. A legegyszerűbb (RS-232, I2C) alapú protokolloktól a legbonyolultabbakig találkozunk óra, órajelszinkronizációval. A két fél nem feltétlenül rendelkezik abszolút időfogalommal, de az átviteli események egyidejűsége szükséges feltétel az átvitel helyességéhez. Az órajelek egyeztetése történhet órajel továbbítással (szinkron átvitel), vagy rekonstrukcióval (aszinkron átvitel). Ilyen terület például a járművek irányítástechnikája is, a mai modern repülőgépek fly-by-wire rendszereinek (MIL-STD-1553, AFDX – az amerikai katonai, valamint a polgári repülésben használt vezérlő hálózatok) egyik kulcspontja az eszközök egyidejű feladatvégzésének biztosítása. Az autóiparban használt CAN busz kiegészítése (TT-CAN) valamint az utódjának szánt FlexRay protokoll szintén tartalmaz órajelszinkronizációs elemeket.

Az óraszinkronizációt szabványosító protokollokra már a ’60-as években születtek szabványok, ilyen például a mai napig ipari irányítási rendszerekben alkalmazott IRIG időkód formátum. Az elektronikus (kvarc) órák elterjedésével lehetővé vált, hogy azok a pontos időt központi adók adatai alapján határozzák meg. Erre szolgál például a Németországból sugárzott DCF77-es protokoll, vagy az amerikai kontinensen népszerű (IRIG kód alapú) WWVB. Ezen protokollok megadták a lehetőséget az emberi érzékelés határain belüli pontossággal szinkronizálni az egyes órákat, az éterben való terjedés minden bizonytalanságával együtt. Az egyik legpontosabb megvalósítása a rádión át szolgáltatott időszinkronizációs szolgáltatásoknak a GPS által szolgáltatott óra, mely segítségével akár ppm hibával vagyunk képesek az UTC-hez kötni az órák futását. Minden helymeghatározási probléma visszavezethető egy időmérési problémára, a GPS által elérhető pontosság kiaknázásához pedig elengedhetetlen, hogy az földi vevők órája mikroszekundumnál nagyobb pontossággal rendelkezzen a műholdak fedélzeti óráihoz

viszonyítva. Szemléltetésképp, 1 µs alatt a fény háromszáz métert tesz meg, így ekkora időbizonytalanság esetén a pozíció meghatározásához alkalmazott végtelenül vékony gömbök valójában háromszáz méteres vastagságú gömbhéjakként képzelhetőek el. Ekkor a metszetük is egy tértartomány lesz (az ideális pont helyett), amely még polgári felhasználás eseteinek túlnyomó többségére is alkalmatlan lenne. A vevőkben általában elérhető az a szolgáltatás, hogy a szinkronizált belső órájukat óraforrásként alkalmazhatjuk egyéb eszközök számára. Az előbbiekben felsorolt szabványokra alapozva rendkívül sok más technológia építette fel a saját belső óraszinkronizációs metódusát.

Az in-band szinkronizációs megoldások közül a legrégebben szabványosítottak a DAYTIME, és TIME protokollok (RFC 867, 868) melyek emberi, valamint gép feldolgozásra definiáltak időegyeztetési szolgáltatásokat. Az alapelvük egy rögzített hálózati portra érkezett (bármilyen formátumú) kérésre adott válasz volt, mely egyszerű hibakeresési célokra működőképes elképzelés, azonban semmilyen módon nem veszi figyelembe a számítógépet övező kommunikációs környezetet, így az általuk szolgáltatott idő nem elégséges adatgyűjtő, irányítórendszerekben történő felhasználásra.

A legismertebb nagy múltra visszatekintő protokoll a Network Time Protocol, mely 1985 óta talán a legrégebbi ma is aktívan használt internetes protokoll. Az NTP célja világméretű kiterjedésű hálózatokban (pl. internet) a pontos idő stabil és biztonságos szállítása a végpontok közt. Érdekesség, hogy a program készítője folyamatosan készíti fel a protokollt a bolygóközi időszinkronizáció okozta problémák kezelésére is. Természetesen az érmének két oldala van, habár a megoldás kifinomult algoritmusokkal szűri a bejövő üzeneteket, és méri a különböző késleltetéseket, választja ki a referenciákat, azonban még lokális hálózaton sem lehetséges a néhány milliszekundumos tartomány alá vinni a pontosságát. Ez a számítástechnika nagyon sok területén elégséges szolgáltatási minőség, azonban mérési adatgyűjtő, és szabályozó rendszerekben gyakran nem elégséges.

Érdekes keveredése az in-band és dedikált protokolloknak az TTEthernet óraszinkronizációs megoldása. A TTEthernet egy a meglevő Ethernet hálózatokkal kompatibilis kommunikációs megoldás, ahol a TTEthernetet megvalósító eszközök hard real-time módon tudnak egymással üzenetet cserélni. Ehhez a beépített óraszinkronizáció mellett nagyszámú hibatűrési protokoll, és hibamegelőzési metódus társul, a protokollban definiált üzenetosztályok mellett. A TTEthernetben történő óraszinkronizáció során az eszközök másodpercenként akár ezer alkalommal is egyeztethetik óráikat, ezen üzenetek terjedési idejére (többek közt) offline felső korlátot adhatunk. Érdekesség, hogy a TTEthernet vezérlőkben implementált órák nem abszolút időt tárolnak (például UTC vagy TAI időskála szerint), hanem a zárt rendszerre jellemző ciklusszámlálót, amivel a rendszeren belül rendezik a mérési adatokat. Ahhoz, hogy eseményeket megfeleltessünk az analóg világból vett időpontoknak, további feldolgozási lépésekre van szükségünk.

Precision Time Protocol

A dolgozatban az IEEE 1588:2008 szabvány által leírt protokollra építve mutatok be egy idő és frekvenciaszinkronizációs megoldást [1] [2]. Ahhoz, hogy a rendszerterv és implementációs részletek jól érthetőek legyenek, érdemes kicsit áttekinteni a protokoll működését, és az általa definiált szerepeket.

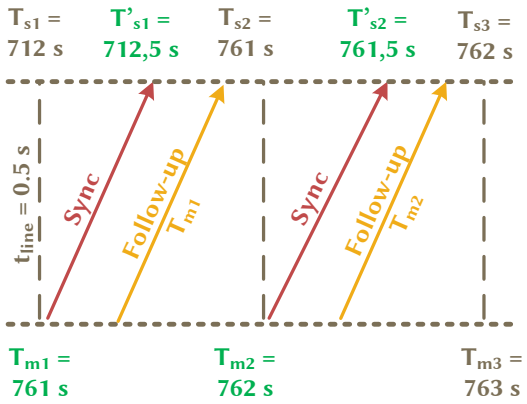
A PTP első kiadása az IEEE 1588-as szabványban jelent meg 2002-ben. Az itt definiált működés javarészt átemelésre került a szabvány 2008-ban kiadott jelenlegi változatába. A PTP

egy master-slave óraszinkronizációs protokollt definiál, elég kevés megkötést alkalmazva mind a végpontokra, mind az üzeneteket szállító kommunikációs médiára. A kommunikációs csatornának alkalmasnak kell lennie a multicast alapú üzenettovábbításra, ugyanis az állomások alapértelmezetten UDP felett a 224.0.1.129 címre küldött multicast üzenetekkel kommunikálnak. A szabvány Ethernet szállítási réteg felhasználása esetén lehetőséget ad a nyers Ethernet keretekkel történő kommunikációra (a 2008-as verzió óta), ezt valós implementációkban fel is használják (Audio-Video Bridging Standard, lásd később). Ennek előnye főként beágyazott rendszerekben jelentkezik, ahol pusztán az óraszinkronizáció igényei miatt nem szükséges TCP/IP stacket a rendszerbe illeszteni, ha azt egyéb szükségletek nem igénylik.

Az IEEE 1588:2008 két részre bontja a szinkronizáció menetét. Először az logikai portok közzéteszik a tulajdonságaikat leíró adatstruktúrákat, mint például az óráik karakterisztikáját, az esetlegesen fellelhető referenciaforrásaik típusát, pontosságát. Ezt a csomagot nevezzük *ANNOUNCE* típusú üzenetnek, mely mérete fixen 106 bájt. Minden logikai port fogadja mindegyik óra bejelentését, és egyszerű összehasonlítással eldönti, hogy ők a továbbiakban master, vagy slave üzemmódban működnek tovább. Az összehasonlítás különböző szempontok alapján történik, fontosság szerint ilyen a porthoz rendelt elsődleges prioritás, az óra pontossága, a rendszer referenciájának típusa, az óra varianciája, az óra másodlagos prioritása, valamint legutolsósorban a logikai port egyedi azonosítója. Egyetlen szinkronizációs szegmensben csupán egyetlen óra lehet master, viszont több összefüggő szegmenst tekintve kijelölhető egy grandmaster óra, akitől minden más végpont származtatja az idejét.

Miután a szerepek letisztázódtak, a slave logikai portok üzenetváltásokkal megállapítják a master órától való eltérésüket, és igyekeznek azt minimalizálni. Az üzenetváltások során az üzenetek tartalmazzák a küldésük idejét. Bizonyos típusú üzenetek küldési idejének megállapítása kulcsfontosságú, azonban ennek a metódusa függ a szinkronizációs hálózatban megállapított egy, vagy kétlépéses szinkronizációs beállítástól. A kétlépéses rendszerekben egy a master által elküldött *SYNC* (86 bájt) üzenet megérkezéskor a slave portban eseményt vált ki, de a slave a küldési időt nem az üzenetben található időbélyegből származtatja, hanem a hozzá tartozó *FOLLOW_UP* (86 bájt) üzenetből. Ez úgy történik meg, hogy a SYNC csomag küldésének eseménye a hálózati kártyában a belső órát mintavételezi, és ezt az információt a driver elkezdi továbbítani a felsőbb protokollrétegek felé. Ennek kezelése az operációs rendszerben egy jól átgondolt tervezést igényel, ugyanis az információ és annak keletkezési ideje elválik. Ez felvethet olyan problémákat, hogy az időbélyegek keveredése miatt a PTP-t megvalósító szoftver esetleg rossz információkat ír a FOLLOW_UP üzenetbe, vagy elveszik a csomag időbélyege egy konkurens socket olvasás miatt. Egy kétlépéses rendszer üzenetváltását ábrázolja az 1. ábra. Egylépéses rendszerben a PHY autonóm módon kitölti a SYNC csomag időbélyeg mezőit, és újraszámolja az Ethernet keret, esetleg az UDP csomag ellenőrző összegét is. Ehhez az adapter meg kell, hogy sértse a réteges protokoll stack szerkezetet, azonban a Start of Frame Delimiter adásának idejét (időbélyegzési pont) kizárólag a PHY-ben vagyunk képesek detektálni. Ekkor a FOLLOW_UP üzenet küldése felesleges, így ezt a portok nem is forgalmazzák. A szinkronizáció egy- vagy kétlépcsős mivoltát a SYNC üzenetek hordozzák magukban. Az órán alkalmazott korrekció ekkor:

$$ofszet_{master-slave} = T_{s1} - T_{m1}$$



1. ábra – Üzenetváltás az ofszet meghatározására (kétlépéses szinkronizáció, két periódusa)

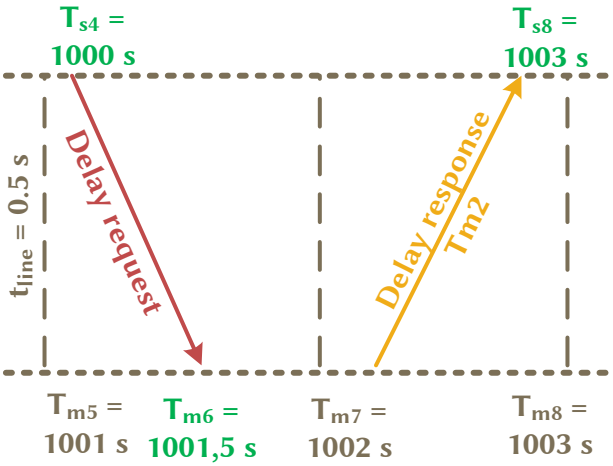
A kezdeti ofszet meghatározása, és korrekciója után a rendszer igyekszik felmérni a kommunikációs hálózat késleltetését (2. ábra). Ehhez a slave logikai portok DELAY_REQUEST csomagokat továbbítanak, és a masterek válaszolnak a slave-ek unicast címeire egy DELAY_RESPONSE üzenettel. Ez az üzenet tartalmazza, hogy a master port saját órája szerint mikor fogadta a kérést. A slave eszköz a kérés elküldésekor feljegyzi a saját órájának állását, majd a két időbélyegből kiszámítja a terjedés idejét. A számítás során feltesszük, hogy a késleltetések szimmetrikusak, valamint az órák a késleltetés mérése előtt már azonos frekvencián üzemeltek. Ebből a mérésből megkapott adatok tehát:

$$ofszet_{slave-master} = T_{m6} - T_{s4}$$

$$késleltetés = \frac{Ofszet_{ms} - Ofszet_{sm}}{2}$$

Az órakorrekció alapjául szolgáló ofszet tehát:

$$ofszet = ofszet_{master-slave} - késleltetés$$



2. ábra – Üzenetváltás a terjedési idő kompenzációja végett

Néha szükségünk lehet arra, hogy nagyobb méretű hálózatokat szinkronizáljuk, vagy különböző szállítási rétegeken is biztosítsuk a szinkronizációs szolgáltatást. Erre a PTP szabvány többféle megoldást nyújt. Nagyméretű hálózatokban a switchek, routerek által kiszolgált egyéb forgalom hatására nem determinisztikus késleltetések jelennek meg a rendszerben, melyek

nagyban csökkentik a szinkronizáció pontosságát. A *Transparent Clock*-ok (TC) segítségével megmérhetjük a szinkronizációs üzenetek tartózkodási idejét a kapcsoló eszközökben, ezáltal az egyik legnagyobb szórású hibát ejthetjük ki a rendszerből. Ha kizárólag ezt a hibát mérjük, akkor beszélünk *End to End* TC-kről. Definiált ezen eszközöknek egy olyan működési módja is, amikor a szomszédos TC-ktől eltelt terjedési időt is mérik, ezen eszközöket nevezzük *Peer to Peer* (P2P) TC-knek. A P2P TC-k alkalmazásával nem csupán a hálózat változó késleltetéséből adódó hibákat küszöbölhetjük ki, hanem például topológia váltáskor minimalizálhatjuk a tranziensjelenségeket. Az itt történő terjedési idő számítása abban tér el a többi végpontokétól, hogy a hiba minimalizálása érdekében egy elküldött *PDelay_Response* üzenet után a *PDelay_Response_Followup* üzenetben továbbításra kerül az előbbi üzenet elküldésének pontos ideje. Ezen eszközök optimális esetben frekvenciaszinkronizált módon működnek a szinkronizációs alhálózaton belül, ezáltal maximalizálhatjuk az elérhető pontosságot.

A *Boundary Clock* (BC) több logikai portot fog össze egyetlen eszközben, és szegmentálja a szinkronizációs hálózatot. Ez azt jelenti, hogy egy portján slave eszközként igazodik egy master port idejéhez, és egy vagy több portján masterként funkcionálva terjeszti a pontos időt. Természetesen fontos, hogy a slave port ugyanazon órát korrigálja, mint amely órából a master portok a saját idejüket származtatják. A Boundary Clock portjai közt PTP forgalom nem haladhat át, a szinkronizációs hálózat számára végpontként funkcionálnak. Ettől azonban egyéb forgalom természetesen áthaladhat rajtuk, az eszköz felépítésétől függően. A BC-k időszinkronizált módon működnek, tehát minden portjuk abszolút időt tart nyilván. Fő felhasználási területük a különböző időskálával üzemelő szegmensek közös grandmasterről történő üzemeltetésének biztosítása. Ez a probléma akkor jön elő, ha egy szinkronizációs alhálózat nem a TAI szerint állapítja meg az időskáláját, hanem egy egyedi folyamatra jellemző nullaponttal rendelkezik, azonban a szegmenseknek azonos frekvencián kell üzemelniük.

Felhasználási példák

Egy jó példa az iparban a nagy pontosságot igénylő feladatra a nagyfeszültségű rendszerek alállomásainak diagnosztikájához szükséges mérési adatok előállítása [2]. Az alállomások diagnosztikája az aktív elemek ki és bemenetein történő áram- és feszültségmérésen alapul. A mérésekből számítható az adott pontot leíró fazor sorozat, és hiba esetén a rögzített fazorokból megállapítható a hiba helye és jellege. A nagypontosságú mérésekhez szükséges időzítési információkat dedikált kábelezésen szolgáltatták egy GPS, IRIG-B, vagy 1PPS kimenetű referenciaóráról. Az adatgyűjtést végző intelligens szenzorok általában IRIG-B bemenettel rendelkeztek, melyet dedikált összeköttetésen keresztül lehetett közvetíteni feléjük. Ezen időzítési információkat hordozó hálózatokat (IEC 61850 Process Bus) nem lehetett kiváltani az adatcserére szolgáló Ethernet alapú megoldásokkal, mivel az NTP, SNTP által szolgáltatott pontosság alatta maradt a hálózati fazorok méréséhez szükséges kívánalmaknak. Ezen az alkalmazási területen a különböző helyen keletkező mérési adatokat akár mikroszekundumonként is sorba kell tudnunk rendezni a probléma megállapítása végett. A dedikált rendszer jelentős kiegészítő kábelezési igényén felül további hiba, hogy egyetlen órakimenet csupán véges számú eszközt volt képes meghajtani, így az elosztás során további plusz eszközök és költségek merültek fel. A feladat által támasztott pontossági követelményeknek megfelel az IEEE 1588-2008 alkalmas implementációja. Ahhoz, hogy az átállás a lehető leggyorsabban megtörténhessen, célszerű úgy továbbfejleszteni az alállomásokat, hogy a különböző szekrények közötti óraszinkronizáció PTP alapon működjön, és csupán a szekrényen belül le-

gyen szükséges az eszközöket az intelligens switch által szolgáltatott IRIG-B jellel szinkronizálni. Ez egy átlagos méretű alállomás esetén több száz méter drága és zavarérzékeny rézvezeték spórolását jelenti. A fentebb ismertetett eszközökkel mind az alállomás vezérlése, mind a diagnosztikája megfelelő módon ellátható. Az alállomások kiterjedése, valamint az átmeneti időszakban igényelt visszafelé kompatibilitás nagyszámú órajelforrásként is alkalmazható Ethernet switch alkalmazását teszi szükségessé.

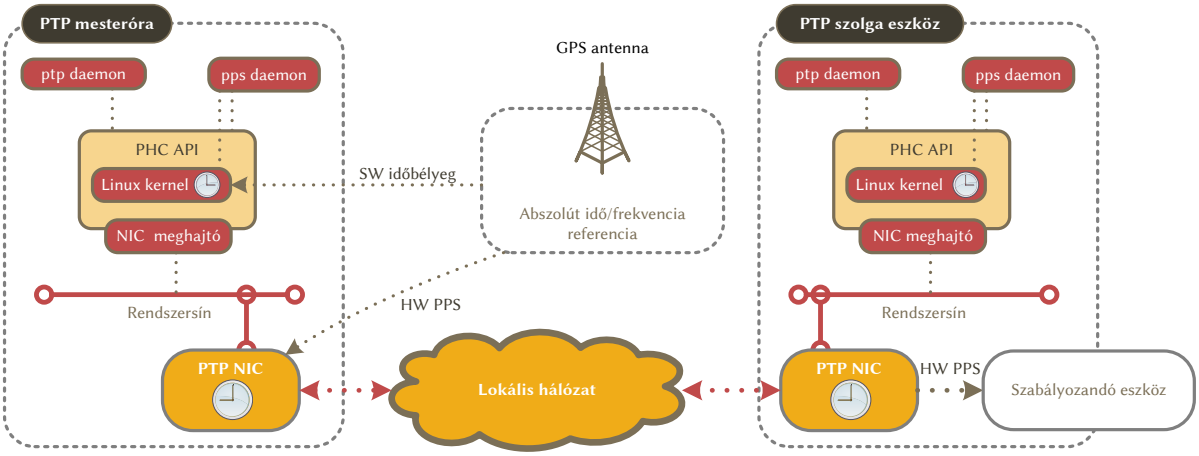
Az Ethernet elterjedéséhez, mint integrált hang és képtovábbító médium a szórakoztatóiparban szükséges, hogy az alapvetően Best Effort működésre tervezett kommunikációs hálózatot további szolgáltatásokkal egészítsük ki. Ezen szabványokat fogja össze az Audio Video Bridging Standard (IEEE 802.1 AVB), mely megoldást nyújt a hagyományos Ethernet hálózat sávszélességfoglalási (802.1Qat), késletetés-menedzsmenttel (802.1Qav) kapcsolatos, és időszinkronizációs (802.1AS) szolgáltatásbeli hiányosságaira [3]. Az Ethernet alapú AV átvitellel megvalósított rendszerek rugalmassága, kalibrálhatósága, zavarérzékenysége jelentősen javul egy olyan rendszeréhez képest, ahol egyetlen központi egységben történik az analóg információ előállítása, és azt analóg kábelezéssel továbbítjuk a végpontok felé, valamint az Ethernet, mint kommunikációs médium nagyobb hatótávolságokkal rendelkezik (100 méter egyetlen szegmens, maximum hét szegmens), mint például a kép és hang egyidejű átvitelére kifejlesztett HDMI (13 méter).

A 802.1AS a PTP egy egyszerűsített profilja, ahol a kommunikáció kizárólag L2 Ethernet felett valósul meg. A szabvány garantálja, hogy maximum 7 szegmens kiterjedésig a szinkronizációs pontosság +- 1 us határon belül marad. A szinkronizáció lépései megegyeznek a korábban ismertetekkel, a cél az, hogy a végpontok órái egy közös master-hez legyenek szinkronizálva. Ez azt is jelenti, hogy a hálózaton nem viszünk át nyers AD/DA órajeleket, hanem a node-ok lokálisan rekonstruálják azt a szinkronizált óráik alapján. Ezen felépítés előnye, hogy különböző frekvenciákon mintavételezett műsorok terjeszthetőek egy időben egyetlen közös médiumon. Ehhez a szinkronizálódó végpontok mellett szükséges az is, hogy a forrás az általa szórt műsort bizonyos időközönként időbélyegekkel lássa el, hogy a lejátszó eszközök lássák, hogy az adott információt milyen óraállásnál kell majd kijátszaniuk. Az így létrejövő elosztott PLL hálózat jittere a nanoszekundum alatti tartományban tartható, ha nagyobb hálózatok esetén a kapcsolóeszközök támogatják a 802.1AVB-t. A különböző hálózati szegmensekben található azonos lejátszási térhez tartozó eszközök késleltetéseinek kezelésére a szabvány kötelező pufferelést ír elő az eszközökben, ezáltal elkerülhető, hogy egy hangforrás eltérő fázisban játssza le a hangot a többi hangforráshoz képest.

Tervezési megfontolások, architektúra

A dolgozatban ismertetésre kerülő rendszer célja egy nagypontosságú időforrás idejének továbbítása osztott kommunikációs médiumon keresztül, kizárólag nyílt forráskódú szoftverelemek használata révén. A megcélzott pontosság legalább 1 µs, melyet hardveres módon PPS kimeneteken keresztül is ellenőrizhetővé kell tennünk. Az architektúra kialakításánál szempont volt, hogy az minél jobban skálázódjon a különböző méretű, felépítésű rendszerek közt, feltéve, hogy azok Linux alapú operációs rendszert futtatnak (3.0 verzió feletti rendszermaggal). A felépítés kialakításánál feltételeztem, hogy nem csupán valósidejű rendszerek számára legyen elérhető a nagy pontosságú működés, ezért azokban jelentkező nem determinisztikus késleltetéseket is kezelni kell. A kiinduló okok azonosak minden esetben, eseményeket kell pontosan időbélyeggel ellátnunk azért, hogy az elosztott eszközök által szolgáltatott adatokat feldolgozó egységek pontosan rekonstruálhassák az események lefolyását egy későbbi időpontban.

A fejezetben bemutatásra kerül egy a 3.0 feletti Linux kernelekben implementált API, mely biztosítja a nagy pontosságú szinkronizációhoz szükséges hardverelemek egységesített felületen történő elérését. Részletesen elemzem ennek a programozói felületnek a kernel és a felhasználói programok felé biztosított kötéseit, és egy későbbi fejezetben demonstrálom egy működő implementációját a felkínált funkciók egy részének.



3. ábra – A tervezett architektúra

Referencia idő/frekvenciaforrás

A rendszerben az abszolút frekvencia referencia szolgáltatja azt az időt és működési frekvenciát, amit a PTP segítségével a meglevő kommunikációs hálózaton továbbítani kívánunk. Az efféle referenciaforrások általában nagyon nehezen illeszthetők be egy meglevő rendszerbe, hiszen méretük (cézium alapú atomórák [1]), vagy elhelyezési igényük (GPS vevők) nagyobb átalakításokat igényelhet a meglevő infrastruktúrában. Szintén nem szerencsés az, hogy ezek az eszközök általában limitált számú referencia kimenettel rendelkeznek, így bizonyos igények közvetlenül nem kielégíthetőek a segítségükkel. Ha nagyszámú eszközt kell referenciaidővel ellátniuk, akkor vagy órajelelosztó hálózatot kell kiépíteni a végpontok, és a referencia között, – amely a skálázhatósága, és költsége nem minden esetben mutat kielégítő ár/érték arányt – vagy osztott kommunikációs médiumon keresztül terjesztik a pontos időt, kis

pontossággal (pl. NTP segítségével). A modern referencia időforrás eszközökben már megtalálható a PTP támogatás, aminek segítségével lokális hálózatokban nagy pontossággal terjeszthetik a hálózaton a referencia időt/frekvenciát.

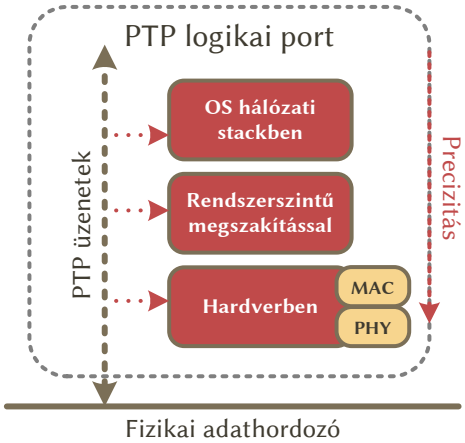
Hálózati csatoló (NIC)

A rendszerben a hálózati csatoló megválasztása alapvetően befolyásolja az elérhető pontosságot, ezért ennek megválasztása kulcskérdés. Egyrészt hardveresen tartalmaznia kell egy hangolható órát, valamint fontos, hogy a beérkező időszinkronizációs csomagokat a hangolható óra segítségével tudja időbélyegezni. Ezen felül természetesen elengedhetetlen, hogy Linux alatt működő driverekkel rendelkezzen, bár ez manapság a vezetékes Ethernet csatolóknál nem szokott nagy problémát jelenteni. Továbbá fontos, hogy implementálva legyen a hardveres időbélyegzés, valamint a driverben a `SIOCSHWTSTAMP` parancs, amivel a hardveres időbélyegzést lehet szabványosan beállítani. Ahhoz, hogy a rendszerbe integrálhassuk, a fedélzeti óra állítását a PHC API-hoz illesztve kell implementálni (lásd alább).

Időbélyegzési pontok

A Linux alapú rendszerekben az **Hiba! A hivatkozási forrás nem található.** által mutatott lehetőségek lehetségesek, röviden összefoglalom mindegyik megoldás előnyét és hátrányát. Általánosságban elmondható, hogy fontos szempont az, hogy a rendszerben miként történik az időbélyegek és csomagok összerendelése, és ezt milyen rendezett adatstruktúrában tárolja az operációs rendszer. A nem megfelelően összerendezett csomag-időbélyeg párosok, valamint az alacsony felbontással tárolt időbélyegek a szabályozási kör instabilitását okozhatják.

Az operációs rendszer hálózati alrendszerének segítségével a rendszeróra felhasználásával gyakorlatilag ma már minden Linux meghajtóval rendelkező hálózati csatoló képes időbélyegeket generálni, azonban ennek a késleltetése és jittere nem determinisztikus, a felbontása a néhány mikroszekundumos tartományba esik, terheléstől függően. A megoldás hátrányai abból fakadnak, hogy a kernel által nyilvántartott szoftveres óra gyakorlatilag egy szokványos bejegyzés a memóriában, melynek elérése, frissítése, és eltárolása egy memória írás-olvasási ciklus mellett gyakran a processzor regisztereinek elérését, az adatok transzformációját igényli, melynek a feldolgozási idejére nem valósidejű rendszerekben felső korlát nem adható.



4. ábra – Lehetséges időbélyegzési pontok ([2] alapján)

A rendszerszintű megszakítás valósidejű rendszerekben egy vállalható kompromisszumokkal rendelkező időbélyegzési szint lehet, de ez nem valósidejű rendszerekben ugyancsak nem al-

kalmazható a megszakítások kiszolgálásának nem determinisztikus mivolta miatt. A beágyazott rendszerekbe történő skálázódással kapcsolatban a dokumentum későbbi részében részletesebben is megvizsgálom ezt a lehetőséget.
A hardverben történő időbélyeg generálást tovább lehet bontani, attól függően, hogy a hálózati csatoló mely részében történik az meg. A MAC-ben történő időbélyegzés egy stabil, megbízható megoldás, hiszen itt már nem egy szoftveres számláló értéke alapján történik a csomagok küldési/fogadási idejének megállapítása, hanem a PTP daemon által kontrollált hardveres óra értékeit használjuk fel. Az itt lehetséges hibaforrások száma már erősen korlátozott. A probléma egyrészt a késleltetésből fakad a PHY és MAC közti MII, SGMII, stb. interfészeken, valamint a MAC már csak bájtszinten látja az adatokat, így az időbélyegzési pontok felbontása csökken, hisz a PHY az adott/vett keret legelső bitjének feldolgozása után képes időbélyeget szolgáltatni.
A fizikai adathordozóhoz legközelebb álló időbélyegzési pont a fizikai réteg vezérlőben (PHY) található. Az Ethernet keret 'Start of Frame Delimiter' mezőjének vétele után a PHY képes a saját órájának állását eltárolni, és ezt közvetíteni a felsőbb rétegek felé. Egy jó implementációban figyelnek arra is, hogy ha a keret sérül, akkor ez az időbélyeg eldobásra kerüljön. Lehetséges, hogy a PHY sokkal nagyobb órajel frekvencián működjön, mint a MAC (125 MHz vs. 625 MHz, SGMII) esetén, ekkor implementációfüggően az időbélyeg felbontása is nagyobb lehet.
A hálózati csatolóban történő időbélyegzésről általánosságban elmondható az, hogy mivel a szinkronizációs csomagok in-band érkeznek a meglevő forgalommal együtt, ezért szükséges egy szűrés, hogy mely csomagok kerüljenek időbélyegzésre, hiszen e nélkül már 100 MBit-en kommunikálva is felesleges többletteher hárulna a hardverre, a feldolgozást végző eszközmeghajtóra, és a PTP daemonra is.

A hálózati csatoló szerepe a mesterórákban

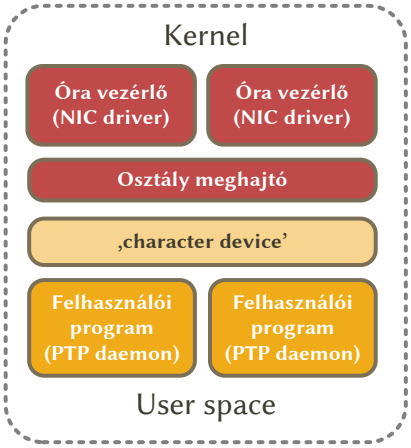
Az általunk megkövetelt pontosságú referenciafrekvencia PTP-n történő terjesztéséhez szükséges azt a hálózati adapter fedélzeti órájában lemásolni, az adapter külső eseményeket mintavételező szolgáltatásának segítségével. Ehhez implementálni kell az eszközmeghajtóban egy nPPS bemenetet, valamint ott, vagy a userspace-ben egy szabályozást megvalósító algoritmust. Ahhoz, hogy a master óra abszolút időt tudjon szolgáltatni, természetesen szükség van egy abszolút időt szolgáltató forrásra is. Ilyen források például a GPS vevők, melyek az 1PPS jel mellett NMEA üzenetekben közvetítik az élekhez tartozó másodperceket is.

A hálózati csatoló szerepe a slave eszközökben

A slave eszközökben cél a saját óra szinkronizációján felül az, hogy nagypontosságú nPPS kimenetet szolgáltathassunk a nem PTP protokollal kommunikáló szinkronizációt igénylő eszközök számára. Ehhez egyrészt a hardver támogatása szükséges, hogy a szinkronizált hardver óra alapján generálhassuk a kimenetet, valamint természetesen szoftver (driver) oldalon is implementálni kell ezeket a funkciókat. Itt is elmondható, hogy az élekhez tartozó abszolút időértékek továbbítását el lehet végezni a PPS daemon segítségével.

PHC API

Ez a programozói felület egy egységes felületet kínál a hálózati kártyák eszközmeghajtói és a szinkronizációs daemonok számára a modern hálózati csatolókban található hangolható órák szolgáltatásainak egységes kihasználására [3].



5. ábra – A PHC API felépítése([3] alapján)

Az első cikk 2010-ben jelent meg a felületről, a felépítését a 5. ábra mutatja. A mainline kernelben 2011 júliusában került beolvasztásra (a 3.0.0 kiadásban). A felhasználói programok a NIC hardverében implementált funkciókat egy karakteres eszközön („character device[1]") át érik el. Ennek segítségével a már meglevő POSIX kompatibilis órakezelő eljárások révén kezelhetjük a hardver órát, az azokban megvalósított funkcionalitás erejéig, például lekérdezhetjük, valamint származtathatunk belőle például számlálókat, időzítőket. A megvalósítás magával hozza azt az előnyt, hogy a karakteres eszközök kezelésére nagyon jól bevált interfészeket használhatjuk (ilyenek például az `open()`, `read()`, `poll()`, `ioctl()` rendszerhívások), melyek logikailag is egyszerűen illeszthetőek a kívánt funkcionalitáshoz – például az óra aktuális értékének kiolvasása egy egyszerű `read()` paranccsal lehetséges. Amennyiben a kívánt funkció nincs lefedve a hagyományos órakezelő eljárások által, azt a karakteres eszköznek küldött `ioctl()`-en át érhetjük el.

A karakteres eszköz a kernellel egy osztálymeghajtóval van összekötve, amely a szabványos hívásokat nyújtja az felhasználói programok felé. Az interfészek implementálásáról az különböző órák eszközmeghajtóját fejlesztőknek kell gondoskodniuk.

A környezet két fő részre osztja az általa kezelt órák szolgáltatásait – a kötelezően implementálandók között van

- az óra lekérdezése,
- az óra beállítása egy megadott időpontra,
- az óra átállítása egy megadott ofszettel,
- valamint az óra frekvenciájának változtatása.

Ezen felül lehetséges kiegészítő funkciókat is implementálni, amennyiben a hardver támogatja ezeket, ilyen többek közt

- a periodikus kimenetek kezelése,
- egyszerű időzítők létrehozása,

[1] A karakteres eszköz egy olyan eszköz, amely bájtos hozzáférést enged a fizikai eszközhöz. A fő különbség a blokkos eszközökhöz képest, hogy általában nem lehetséges az eszköz által szolgáltatott adatfolyamban visszafelé haladni.

- a PPS jelek, események biztosítása,
- külső események időbélyegzése.

A kiegészítő szolgáltatások implementálásával egy nagypontosságú, teljes értékű referenciaórát állíthatunk elő abban az esetben, ha a felhasznált hálózati csatoló hardveres órája, és időbélyegzési szolgáltatásai megfelelnek a minőségi kívánalmainknak.

PTP daemon

A PTP daemon feladata az IEEE 1588:2008 szabványban rögzített algoritmusok implementálása, a logikai portok kezelése és a szabvány implementációjához szükséges, de részleteiben nem definiált szolgáltatások megvalósítása – ilyen például az ofszet minimalizálása, amit az órák hangolásával tehetünk meg. A daemon kiválasztását egyszerű szempontok határozzák meg, egyrészt Linux alatt működőképesnek kell lennie, valamint együtt kell működnie a PHC API szolgáltatásaival. Ennek a nyílt forráskódú alternatívák közül a *linuxptp* projekt tesz eleget. A projekt fejlesztői a korábbi munkámban bemutatott ptpd2 daemonnak is elkészítették egy forkját, azonban az már nem támogatott, az internetről el is távolították.

Hasznos, ha a daemon paraméterei futás közben hangolhatóak, így újraindítás nélkül lehetséges például az óra karakterisztikáját leíró adatmezők frissítése, ezáltal a hálózat automatikus újrakonfigurációjának biztosítása. Ehhez a linuxptp projekt biztosít segédprogramot.

PPS daemon

A PTP daemon ebben a megoldásban elsődlegesen a hálózati csatolón található órát igazítja a mesteróra idejéhez, azonban előfordulhat, hogy a kernelben is szükségünk van az elérhető legpontosabb időszinkronizációra. Szélsőséges esetben az is elképzelhető, hogy a szabályozási kört megvalósító algoritmus pontossága csorbát szenved a nem megfelelő stabilitású rendszeridő miatt. A PHC API egy könnyen elérhető felületet kínál az eszközmeghajtó programozók számára a kernel meglevő PPS szinkronizációs infrastruktúrájához történő kapcsolódáshoz.

A rendszeróra szinkronizációja két részre bontható, egyfelől a drivernek regisztrálnia kell a PPS jeleket kezelő rutinját, és ezt az osztálymeghajtó automatikusan beregisztrálja a kernel PPS forrásai közé. Ezután a PTP szinkronizáció stabil fázisában (ahol már csak frekvenciakorrekció történik) egy felhasználói program (ami lehet egy különálló segédprogram, vagy a PTP daemon állapotgépe) engedélyezheti a PPS események kibocsátását, melyet a PPS daemon olvasni tud, és ennek megfelelően tudja szinkronizálni a rendszeridőt.

Az előzőekben említett linuxptp projekt tartalmaz a rendszeróra szinkronizációjához szükséges PPS klienst, valamint fejlesztettem egy a hálózati kártyán levő PPS forrást engedélyező segédprogramot.

Szerepe a mesterórákban

A PPS daemon segítségével kérdezhetjük le a referencia időforrás abszolút idejét. Mivel ez alapvetően tisztán szoftveres úton valósul meg, ezért szükségünk van a referencia frekvencia pontos követéséhez hardveres kiegészítő szolgáltatásokra (erről bővebben a **Hiba! A hivatkozási forrás nem található.** alpontban írok). A Linux kernelben úgy alakították ki a PPS alrendszert, hogy több független PPS forrás is regisztrálható, és a programok dönthetik el, hogy melyiket használják fel a szinkronizációhoz. A hardveres óra jelenlétekor a rendszer egy az idő-

bélyegekkel történő interpolációnál pontosabb PPS forráshoz jut, amely növeli a rendszeróra pontosságát.

Szerepe a slave eszközökben

A slave eszközökben a PPS daemon feladata megvalósítani a rendszeróra szinkronizációját, valamint igény szerint tovább szolgáltatni az abszolút időt a rendszer legacy kimenetein (pl. timestring-et, NMEA string-et a soros portokon). Amennyiben a rendszeridőt is szinkronizáljuk a hardveróra mellett, úgy *kétszintű óraszinkronizációról* beszélünk – egyrészt a lokális hálózati csatoló óráját hangoljuk nagy pontossággal a PTP hálózaton keresztül, másrészt a PPS daemon egy független szabályozási kört biztosítva beállítja a rendszerórát a hardverórához igazítva.

Lokális hálózat

A Precision Time Protocol alfejezetben ismertetésre került, hogy a PTP alapvetően egy lokális hálózatokban történő óraszinkronizációra szánt protokollt definiál. Egy lokális hálózat lehet például egy épületen belüli Ethernet infrastruktúra. Ekkor a hálózatról elmondható, hogy előre ismert a jellemző viselkedése (jellemző kihasználtság, késleltetések, stb.), így például a szabályozó körök paramétereinek hangolásánál ezek figyelembe vehetőek. Természetesen nem lehet elégszer hangsúlyozni, hogy az Ethernet nem determinisztikus a késleltetéseit tekintve, de a PTP daemonok ezért tartalmaznak a terjedési idő számítására algoritmusokat, valamint a kritikus pontokban lehetséges Boundary Clock-ok elhelyezésével szegmentálni az óraszinkronizációs hálózatot, vagy Transparent Clockokkal kompenzálni a nem becsülhető késleltetéseket.

Mérési elrendezés, a megvalósítás jelenlegi fázisa

Ebben a fejezetben ismertetem az általam elvégzett implementációt részleteiben, megvizsgálom a megvalósítási kérdések döntési pontjait, és kijelölöm a továbblépési irányokat. A jelenlegi mérési elrendezést részletesen a **Hiba! A hivatkozási forrás nem található.** mutatja be, így először a működő rendszer komponenseit vizsgálom meg. A rendszert alkotó nagyobb eszközök mind egy-egy a fentiekben leírt komponensnek feleltethetőek meg, így a fejezet felépítése is eme komponensek szerint van szétbontva.

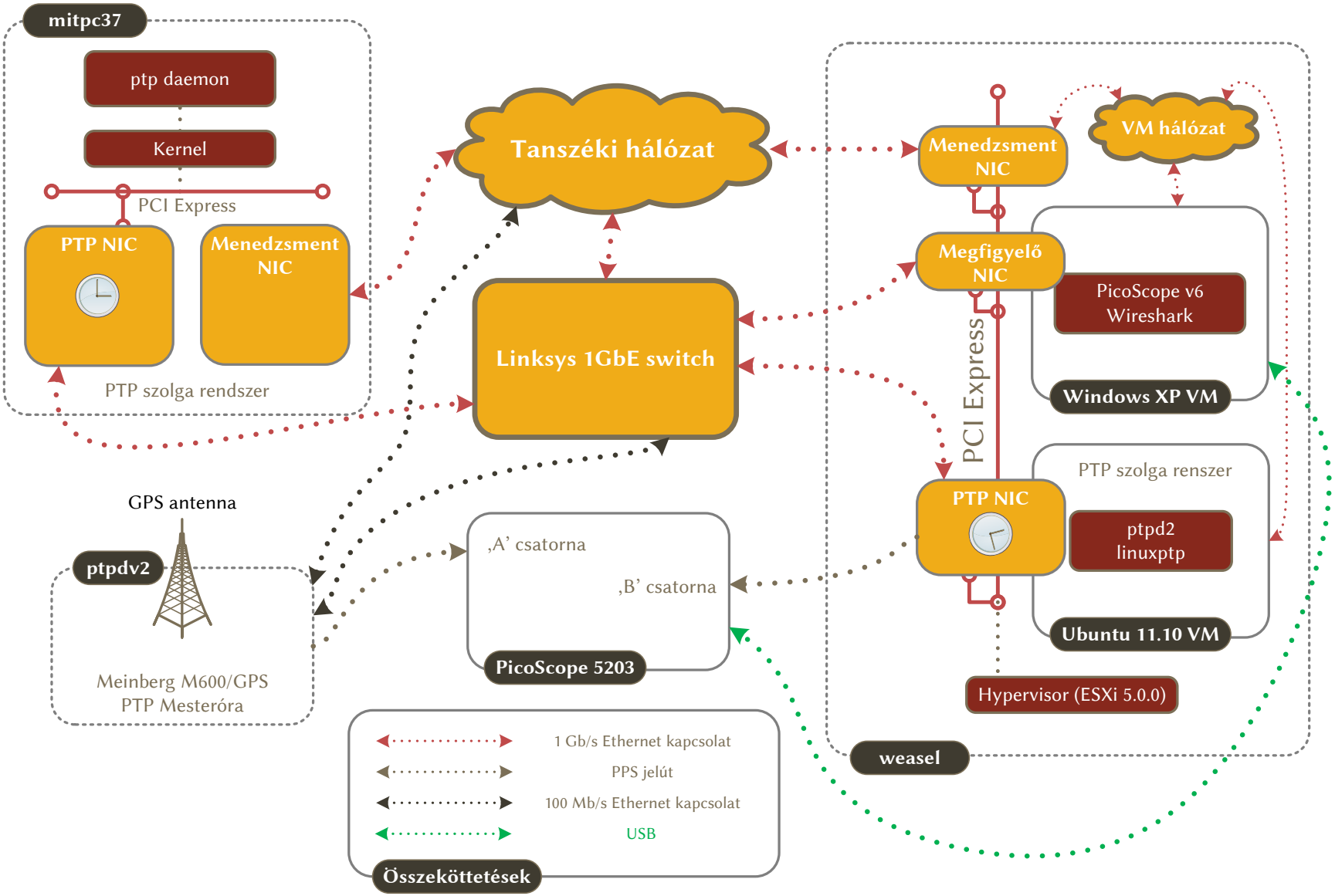
A választott szinkronizációs végpontok nem real-time PC alapú rendszerek. Választásom indoka az, hogy egyrészt demonstrálni szeretném, hogy in-band kommunikáció segítségével nem valósidejű környezetben is lehetséges nagy pontosságú idő és frekvenciaátvitel, valamint ezáltal olyan fejlesztőkörnyezethez jutottam, amelyben korábbi ismereteimmel egyszerűen boldogulok, és kényelmesen fejleszthetek rá a tanszéken kívülről is.

Linksys switch – lokális hálózat

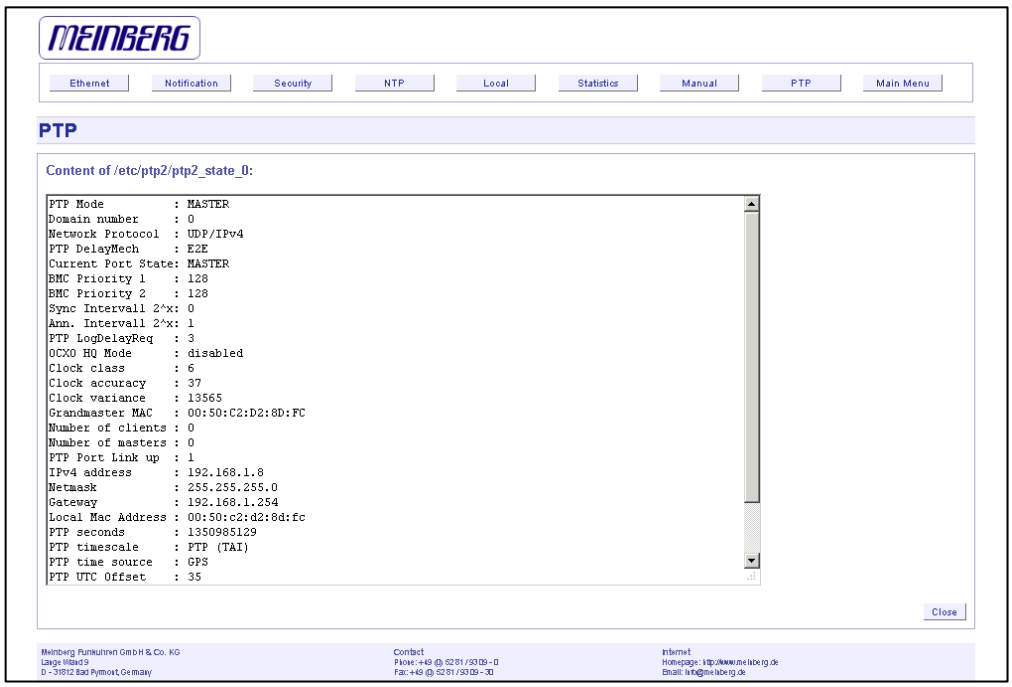
A mérések során a szinkronizálódó végpontok egy Linksys SRW2008-as típusú 1000BASE-T Gigabit Ethernet szabványt támogató 8 portos switchen keresztül voltak összeköttetésben. A switchen két VLAN volt beállítva, az egyik az uplink portnak, ezen keresztül volt elérhető a menedzsment felülete a tanszéki hálózatból. A második VLAN-ba a maradék 7 portja tartozott, melyek egyetlen közös szinkronizációs hálózatot alkottak. UDP/IP feletti kommunikáció esetén a 192.168.1.0/24-es IP tartomány volt az eszközök számára kiosztva. A switch egy egyszerű store-and-forward architektúrájú 1000BASE-T portokkal rendelkező switch, mindennemű PTP támogatás nélkül. A Boundary Clock, Transparent Clock funkcionalitás hiányát a mérési eredmények kiértékelésénél fogom részletesebben elemezni.

ptpdv2 – PTP mesteróra

Az előzetesen tárgyalt rendszertervben látható PTP mesteróra a jelenlegi állás szerint egy dedikált mesteróra (Meinberg LANTIME M600/GPS/PTP), mely egyetlen 1U rack méretű eszközben tartalmaz minden szükséges eszközt a stratum 1-es NTP, valamint PTP kiszolgálóként való működéshez [4]. Az eszköz egyrészt az egyik Ethernet portján közvetlenül csatlakozik a tanszéki hálózathoz, ahol webes/ssh felületen keresztül lehetséges a konfigurációja. A mesteróra PTP hálózati portja a kétlépcsős szinkronizációt támogatja, a lokális hálózatban a 192.168.1.8-as IP címen elérhető, 100BASE-TX szabványú kommunikációra képes. A mesteróra egyedi heterogén architektúrájú alkotóelemekből épül fel (x86, ARM, FPGA), és egy beágyazott 2.6.15-ös verziószámú Linux kernelre van ráépítve a szoftver stackje.



6. ábra - A mérési elrendezés



7. ábra - PTP adatok az M600 mesterórán

A mesterórához csatlakozik egy GPS antenna, ez referenciaforrásként szolgál az aktuális abszolút időről, és frekvenciáról. Az antenna egy fontos tulajdonsága, hogy belül lekeveri a gigahertzes tartományban levő nyers GPS jelet (L1 kód – 1575,2 MHz helyett 35,4 MHz-re), ezáltal olcsó RG58-as kábelen is továbbítható épületen belül a jel, akár 300 méteres távolságra is. Ez nagymértékben csökkenti az installálási költségeket, hiszen a nagyfrekvenciás kábelek beszerzési árai, és beszerelési költségei is magasabbak.Az eszközön található nagymennyiségű legacy óraszinkronizációs kimenet is, ilyenek például:

- RS232 output, serial interface, time telegram
- Error relays output
- 10MHz output
- Pulse Per Second output
- Pulse Per Minute output
- Frequency synthesizer output
- Time code output - AM (modulated)
- Time code output - DCLS (unmodulated)

A ’Frequency synthesizer’ kimenet segítségével a felhasználó állíthatja be a kívánt jelalakot, és frekvenciát. Az általam használt PPS kimenet egy 20%-os kitöltési tényezőjű TTL jelszintű 1Hz-es jelet generál, melynek a eltérése maximum +/- 20 ns a GPS időalap másodperceitől (feltételezve, hogy az eszköz épp a GPS-hez szinkronizálva van).

A mesteróra ezen felül el van látva egy nagy stabilitású hőmérsékletstabilizált oszcillátorral (OCXO), melyre a gyártó az alábbi adatokat adta meg:

1. táblázat - OCXO HQ adatok

rövid idejű stabilitás (1s):	0,005 ppb
PPS pontossága	+-100,0 ns
holdover hiba (24 h)	+-22,0 us

Ezen adatok akkor fontosak, ha a rendszerünket fel kívánjuk készíteni a referencia időalap elvesztésével járó eseményekre – a jó minőségű oszcillátor segítségével a GPS jel rövid idejű kiesésével még mindig a kívánt minőségi paramétereken belül maradhatunk, bár egy nap alatt már kívül esünk.

weasel – PTP slave host eszköz, fejlesztőrendszer

A Meinberg órához, és a hálózati kártyákhoz hasonlóan ehhez a fejlesztőrendszerhez is az Intel felajánlásából jutottam hozzá. A fejlesztőrendszer egy HP Proliant ML150 G5-ös szerver, mely jelenlegi kiépítésében egy foglalatban tartalmaz egy Intel Xeon E5504-es processzort, valamint 6 gigabájt RAM-ot. A rendszer bővíthetősége kiváló, 1 darab 16x-os, 2 darab 8x-os PCI-Express v2, és 1 darab 8x-os PCI-Express v1 bővítősín található benne. Jelenleg egy Intel i350-T4 4 csatolós szerver adapter található benne, melynek két portja van kiosztva a két VM között.

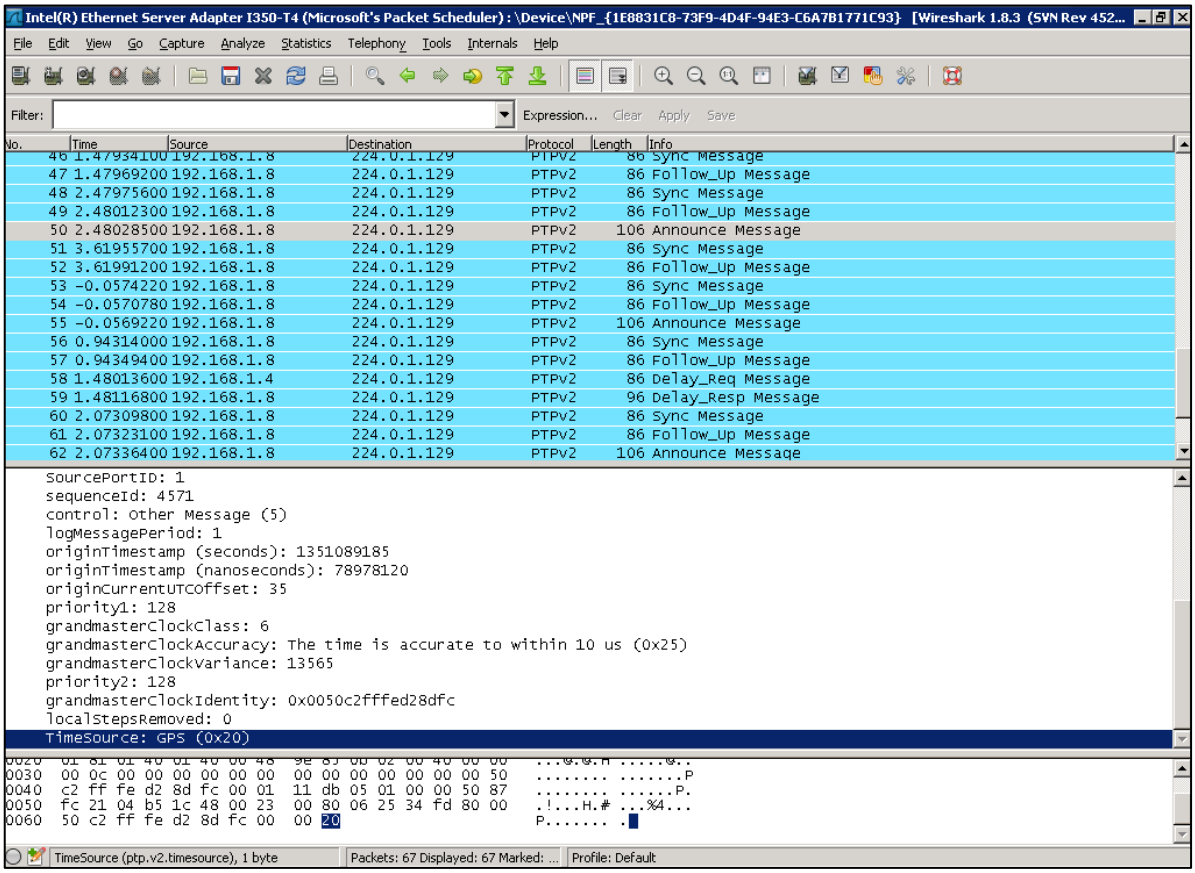
A kezdetek kezdetén ezt a gépet is úgy használtam, mint a mitpc37 fejlesztőrendszert, azaz egy Linux futott rajta, és távoli kapcsolaton keresztül tudtam a kódot fejleszteni a kernelbe. Mivel ez egy kisvállalati szerver szerepkörbe szánt hardver, ezért található rajta egy szervizprocesszor, amit saját IP-jén elérve a lefagyott rendszer újraindítására tudtam használni (a kernel fejlesztés során gyakran megesik, hogy egy hibás címre írás, vagy olvasás kernel panic-ot okoz). Ez megóvott sok kellemetlenségtől a többi fejlesztőrendszerhez képest, azonban a hardver nagyrészt kihasználatlan volt.

A nyár folyamán döntöttem úgy, hogy a gyors verzióváltásokhoz, rugalmas hardverkiosztáshoz elengedhetetlen egy bare-metal architektúrájú Hypervisor feltelepítése. A választásom a VMware ESXi 5.0.0-re esett, erre installáltam fel a fejlesztési környezetet adó virtuális gépeket. Egy m0n0wall virtuális gép osztja tovább a tanszéki hálózatot a virtuális gépek számára a 192.168.2.0/24 címtartományban. A fizikailag létező hardver nagyszámú hálózati kártyát fogadthat egyszerre, azonban a processzor VT-d, és a hypervisor hardver bypassing (DirectPatch I/O) támogatásának köszönhetően ezeket át lehet adni közvetlenül a virtuális gépek kezelésébe, így a rajtuk fejlesztett driverek közvetlenül érik el a hardverek regisztereit. Az így nyert rugalmasság sokrétű, egyrészt a hagyományos módon fejleszthetem az eszközmeghajtókat, de a kártyákon található portokat egyedileg rendelhetem minden VM-hez, ezáltal könnyen kezelhető fejlesztőkörnyezeteket kapok – például minden VM csak egyetlen hálózati adaptert lát, így egyértelműek az eszköznevek, a scripteket könnyű migrálni köztük. Amennyiben a betöltött meghajtó hibásan viselkedne, csupán az azt betöltő VM keletkezne kernel panic, hiszen az általa hardver, és az azt vezérlő kernel teljes mértékben el van szeparálva a gépen futó többi szolgáltatástól [7].

Windows XP VM

Ennek a VM-nek két szerepe van a mérési elrendezésben, egyrészt a hozzáadott i350-es adapteren keresztül meg tudja figyelni a hálózati forgalmat (IP: 192.168.1.222), másrészt ezen fut a hardveres kimenetek megfigyeléséhez szükséges PicoScope szoftvere. A hálózati forgalom megfigyelésére a Wireshark Windows portját használom (pl. 8. ábra). A rendszert távolról RDP-n át adminisztrálom, a keletkező mérési adatokat pedig Dropbox segítségével osztom meg az elsődleges fejlesztő számítógépemmel. A PicoScope szoftverének segítségével a hardveres kimeneteknek olyan tulajdonságait tudom megmérni, mint az átlagos eltérés a mesteróra PPS

jelétől, valamint az eltérés szórása, valamint képes digitális foszfor üzemmódban szemábrákat készíteni a négyszögjelekről.



8. ábra - a mesteróra hirdetett pontossága Wiresharkban megvizsgálva

Ubuntu 11.10 VM

Ennek a virtuális gépnek a célja a fő fejlesztőeszközként működés, melynek biztosítására egy i350-es portot adtam át közvetlen kezelésre (IP: 192.168.1.4), 1 gigabájt RAM-mal egyetemben. Az általa futtatott Linux kernel a 3.0.0 verziószámú (x64 SMP), újrafordítva a szükséges PHC API támogatással. A kernel fordításának részletes menetét a [7]írja le. A fejlesztéshez kiindulási pontként használt igb driver a 4.0.17-es változat volt, mivel itt már szétválasztották a hálózati kártya normál működését biztosító kódot az időszinkronizációval kapcsolatos kódtól. Fontos megjegyezni, hogy a 3.2.9-es verzióhoz jómagam is fejlesztettem hardverórát kezelő drivert, mind a PHC API felhasználásával, mind a nélkül, azonban a PHC API-sat nem éri meg karbantartani párhuzamosan a gyári driverek mellett (melynek időszinkronizációs kódbázisát a linuxptp daemon egyik fejlesztője tartja karban), az API nélkülit pedig csupán proof-of-concept meghajtóként használtam fel, hogy 2.6-os kernelverziók alatt is lehetséges a precíziós órakimenet létrehozása. A hálózati csatoló képes a bejövő csomagok szűrésére időbélyegzést (is) tekintve, valamint minden kimenő csomagot időbélyegez, amennyiben az a 319-es UDP porton kerül kiküldésre, vagy a PTP Ethertype-jű Ethernet keret.

Linuxptp beüzemelése

Mint azt már említettem, egyedül a Linuxptp daemon felelt meg a szerepben támasztott követelményeknek, ezért röviden jellemzem az integrációját a rendszerben. A daemon beszerezhető a linuxptp.sf.net honlapról, telepítése a git repository klónozásával kezdődik:

```
$ git clone git://git.code.sf.net/p/linuxptp/code linuxptp
```

Ezután a linuxptp könyvtárban kiadott make parancs segítségével a forráskód lefordítható. A fordításhoz szükséges, hogy a fordító környezet definiálja a PHC API függvényeit, ha ez a támogatás nincs bekapcsolva a kernelben, akkor a linkelés során figyelmeztetést kapunk, és a kód nem fog megfelelően működni! A könyvtárban több bináris fájl keletkezik, először a szinkronizációt megvalósító ptp4l szoftver beállítását mutatom be.

A programnak root jogok szükségesek a futáshoz, hiszen közvetlenül éri el a hálózati és óraeszközt. A szolgáltatás elindítása az alábbi módon lehetséges:

```
$ sudo ./ptp4l -i eth1 -p /dev/ptp0 -v
```

A –v kapcsoló elhagyható, pusztán tesztelési célokból szükséges (az stdout-ra irányítja kimenetet), az üzenetek alapesetben a syslog-ba kerülnek. A szoftver beállításait a -? kapcsolóval lehet megjeleníteni, a gyakran használt beállításokat pedig konfigurációs fájlokba menthetjük, melyeket az –f kapcsolóval tölthetünk be. A mérési eredményeknél majd láthatjuk, hogy a szoftver megfelelt a vele szemben támasztott követelményeknek.

Példa kimenet:

```
ptp4l[506577.815]: selected /dev/ptp0 as PTP clock
ptp4l[506577.815]:  failed  to  read  out  the  clock  frequency  adjustment:
Operation not supported
ptp4l[506577.815]: port 1: get_ts_info not supported
ptp4l[506577.816]: driver changed our HWTSTAMP options
ptp4l[506577.816]: tx_type   1 not 1
ptp4l[506577.816]: rx_filter 1 not 12
ptp4l[506577.816]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[506577.817]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[506578.433]: port 1: new foreign master 0050c2.fffe.d28dfc-1
ptp4l[506582.613]: selected best master clock 0050c2.fffe.d28dfc
ptp4l[506582.613]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[506583.682]: port 1: minimum delay request interval 2^3
ptp4l[506584.773]:  master  offset  -52357562843  s0  adj         +0  path  delay
6795
ptp4l[506585.772]:  master  offset  -52357567251  s0  adj         +0  path  delay
6795
ptp4l[506586.923]:  master  offset  -52357572379  s0  adj         +0  path  delay
6795
ptp4l[506587.943]:  master  offset  -52357576915  s1  adj         +0  path  delay
6795
ptp4l[506588.943]:  master  offset          -8016  s2  adj     -8016  path  delay
6795
ptp4l[506588.943]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[506590.083]:  master  offset          -3964  s2  adj     -6369  path  delay
6795
ptp4l[506591.093]:  master  offset          -1993  s2  adj     -5587  path  delay
6795
ptp4l[506592.093]:  master  offset           -848  s2  adj     -5040  path  delay
6795
ptp4l[506593.223]:  master  offset           -198  s2  adj     -4644  path  delay
6795
ptp4l[506594.243]:  master  offset             24  s2  adj     -4482  path  delay
6795
```

```
ptp4l[506595.243]: master offset            51 s2 adj    -4448 path delay
6795
ptp4l[506596.343]: master offset            30 s2 adj    -4453 path delay
6795
ptp4l[506597.363]: master offset            44 s2 adj    -4430 path delay
6795
ptp4l[506598.363]: master offset           104 s2 adj    -4357 path delay
6703
ptp4l[506599.493]: master offset            18 s2 adj    -4412 path delay
6703
ptp4l[506600.513]: master offset           -32 s2 adj    -4456 path delay
6703
ptp4l[506601.513]: master offset           -27 s2 adj    -4461 path delay
6703
ptp4l[506602.654]: master offset           -24 s2 adj    -4466 path delay
6703
ptp4l[506603.663]: master offset           -21 s2 adj    -4470 path delay
6703
ptp4l[506604.663]: master offset             2 s2 adj    -4454 path delay
6703
ptp4l[506605.803]: master offset            29 s2 adj    -4426 path delay
6703
```

A PTP daemon mellett még két segédprogramot használtam fel a megvalósításhoz. A phc2sys segédprogram a kernel óráját szinkronizálja PPS forrásokhoz. Használata a

```
$ sudo ./phc2sys –p /dev/ppsx
```

paranccsal lehetséges, ahol az ’x’ a kívánt PPS forrás sorszáma. A program működéséhez szükséges, hogy a rendszerben legyen aktív PPS forrás, ezt a rendszerben a hálózati kártya segítségével valósítom meg. A program kimenetén meg tudjuk figyelni a eltérés mértékét, a szabályozási kör állapotát, és az PPS trigger keletkezésének pontos időpontját is.

```
pps -394706209 s1 1351443897.605293791 drift 0.00
pps -394693632 s2 1351443898.605306368 drift 0.00
pps -394712501 s3 1351443899.605287499 drift 0.00
pps -394692576 s4 1351443900.605307424 drift 4544.31
pps    -22043 s4 1351443901.999977957 drift -2068.59
pps     -1889 s4 1351443902.999998111 drift -2635.29
pps     75009 s4 1351443904.000075009 drift 19867.41
pps    -68642 s4 1351443904.999931358 drift -725.19
pps      2364 s4 1351443906.000002364 drift -15.99
pps     16000 s4 1351443907.000016000 drift 4784.01
pps     15239 s4 1351443908.000015239 drift 9355.71
pps    -32807 s4 1351443908.999967193 drift -486.39
pps      4854 s4 1351443910.000004854 drift 969.81
```

Az NPPS periodikus kimenet beállításához egy saját segédprogramot készítettem ’perpps’ néven. A program két funkciót valósít meg a kártyán, egyrészt bekapcsolja az 1PPS kimenetet, másrészt NPPS kimenetet tud aktiválni. Ezt a kártyát vezérlő driverben elvégzett implementáció miatt választottam ketté, mivel ott az 1PPS kimenet aktiválása a PTP_CLK_REQ_PPS, míg a nagyfrekvenciás kimenet a PTP_CLK_REQ_PEROUT kéréshez volt kötve.

Az Intel i350-T4 hálózati kártya

A rendszerbe illesztett hálózati csatoló a négyportos 1000BASE-T szabványú kommunikációt megvalósító Intel i350 családba tartozó eszköz volt (9. ábra). A kártya a hagyományos

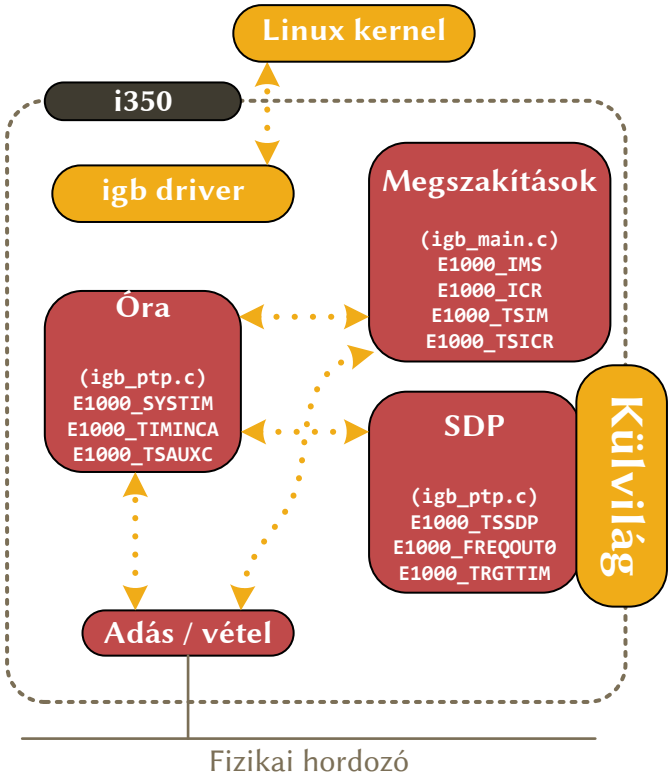
Ethernet kommunikációhoz szükséges funkciókon felül az időszinkronizációhoz is nyújt hardveres támogatást [7]. Ezt egyrészt a PHY-ben megtalálható hardveres időbélyegzés biztosítása, másrészt az adaptereken található hangolható óra megléte. A kártyán 4 darab független hálózati adapter található egyetlen szilíciumon, ami azt jelenti, hogy a portok egymástól független fedélzeti órákkal rendelkeznek, az egymás közti megosztásuk nem lehetséges! Ez egy esetleges Boundary Clock megvalósításban vethet fel megvalósíthatósági kérdéseket, mivel biztosítanunk kell azt, hogy a master és a slave logikai portok órái szinkronizálódjanak a kártyán belül.

A kártyán található kimenetek legnagyobb amplitúdója 3,3 V, és maximum 4 V-os bemenő jelet fogadnak. A kártya kizárólag kétlépéses szinkronizációra képes (lásd: Precision Time Protocol fejezet).

9. ábra – az Intel T350-T4 hálózati kártya egy jellemző kiszerelése

A kártya blokkjai

A kártyán történő óraszinkronizáció megvalósításához a **Hiba! A hivatkozási forrás nem található.** által mutatott blokkokat tartalmazza. Az kártyát az *igb driver* működteti, ez biztosítja a szükséges interfészek implementálását a Linux kernel felé. Ez nagyszámú hálózati API, valamint a PHC API megvalósítását jelenti. A driver a programozó felé több, a fejlesztést megkönnyítő makrót nyújt. Jó példa erre a regiszterek kezelését elősegítő makrókészlet. Az `E1000_READ_REG(hw, regiszternév)` makró segítségével olvashatjuk őket, míg az `E1000_WRITE_REG(hw, regiszternév, érték)` makróval írhatunk beléjük. Gyakran előfordul, hogy több összetartozó írási műveletet is végzünk a driverben, ezért a tényleges írási (buszműveletek) engedélyezésére használhatjuk az `E1000_WRITE_FLUSH(hw)` makrót. Ekkor tárolódnak el ténylegesen a kártyában az beírt értékek. A driver a különböző blokkokhoz tartozó forráskódot elkülönítve kezeli, így az a fejlesztés során átlátható marad. A zárójeles fájlnevek tartalmazzák az általam fejlesztett függvénydefiníciók helyét. A regisztereket címző, és bitmezőiket leíró makrók elhelyezése az *e1000_defines.h* és *e1000_regs.h* fájlban történt meg.



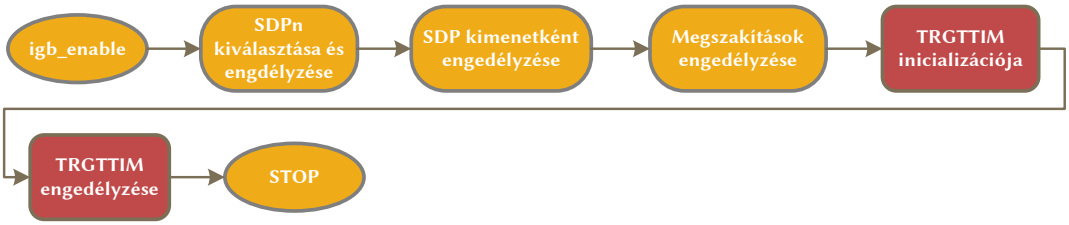
10. ábra – az i350-es kártya időszinkronizációs blokkdiagramja

Az i350-es adapterek óraábrázolása effektív 72 bit-en történik, regisztercsoportok segítségével. A SYSTIMR regiszterben 32-biten tároljuk az időt 2^-32 ns időegységekben, a SYSTIML regiszter ugyancsak 32 biten ábrázolja 1 ns felbontással, míg a SYSTIMH 8 biten ábrázolja 2^32 ns felbontással. Ez azt jelenti, hogy az adapterek fedélzeti órái kevesebb, mint 20 perc alatt átfordulnak, ezért alkalmatlanok arra, hogy közvetlenül az abszolút időt tároljuk bennük. Ennek megoldására a Linux kernel kínál egy felületet (timecounter struktúrák), mellyel meg lehet feleltetni egymásnak az adapter lokális óráját, és a valós időt. Egy fontos megkötés van, az adapter órája maximum egyszer fordulhat át két transzformáció közt, különben az összerendelés nem lesz megfelelő. A rendszeridő az adapter 125MHz-es órájából származtatott 8 ns-os felbontással bír, és az alábbi képlet alapján frissül:

$$\mathrm{SYSTIM}[k+1] = \mathrm{SYSTIM}[k] + 8\,\mathrm{ns} + \mathrm{TIMADJ} + \mathrm{TIMINCA}$$

ahol a TIMADJ egy 40 bit széles regisztercsoport, és előjelesen tárolja, hogy a következő óraciklusban mennyit kell módosítani a rendszeridő értékén (majd törlődik az értéke), a TIMINCA pedig ugyanezt az információt tartalmazza 32 biten 2^-32 ns felbontással (és értéke a következő írásig permanens). Az 1PPS kimenet előállítására felhasznált TRGTTIM regisztercsoportok szintén 40 bites effektív szélességgel bírnak.

A hardveres kimenet algoritmusa



11. ábra - PPS kimenet inicializációja IT alapon

A 11. ábra azt ábrázolja, hogy milyen folyamat szerint inicializálom az 1PPS kimenetet. Az `igb_enable()` egy az osztálydriver által nyújtott interfész függvény, melyben az általam megvalósított inicializációs függvény a `PTP_PPS_ENABLE` ioctl hatására fut le.

Az `E1000_TSSDP` regiszter rendeli össze az kártya időszinkronizációs blokkjait a fizikai kimenetekkel (SDP – *Software Definable Pin*). Az `E1000_TS_SDPn_SEL(x)` makróval választom ki a kívánt összerendelést, és az `E1000_TS_SDPn_EN` makró által beállított bit engedélyezi azt. Ezek után az SDP sorszámától függően az `E1000_CTRL` (0,1), vagy `E1000_CTRL_EXT` (2,3) regiszterben beállítom, hogy az adott SDP *kimenet* legyen. Ezek után engedélyezem az `E1000_TSAUXC` regiszterben az *impulzuskimenetet*, ami azt jelenti, hogy az interfészen található kettő target time regiszter által meghatározhatok egy tetszőleges szélességű impulzust. Én azt az esetet választottam, amikor a 0-s jelű regiszterbe írt idő elérése esetén a kimenetre felfutó él kerül, míg az 1-es regiszterbe beírt idő lefutó élt generál (`E1000_TSAUXC_PLSG0`).
Ezek után következik az inicializáció kritikus szakasza, a *pontos target time-ok meghatározása*. A Meinberg mesteróra 20 %-os kitöltésű 1PPS jelet szolgáltat, ezért én is ilyet választottam a megvalósításban. Ez azt jelenti, hogy az `E1000_TRGTTIM[HL]1` regisztercsoport 200 ms-mal nagyobb értéket tartalmaz minden iterációban, mint a 0-s jelű regiszterek. A kezdő él idejének meghatározására az alábbi algoritmust használom:

1. Kiolvasom a kártya órájának transzformálatlan értékét (cc)
2. Ezt áttranszformálom a szoftveres rendszeróra időalapjába, hiszen ott abszolút UTC időt tárolok
3. Kiszámolom egészosztással, hogy mennyi idő telt el a kiolvasott időbélyegben az egész másodperc óta (remainder)
4. Az indulási idő biztonsági okokból 1 másodperccel késleltetve van, azaz
   `start = (cc) + 2 másodperc – remainder – hiba`
   A hibatag a mérési eredmények kiértékelésénél lesz kifejtve, ezt állíthatom a driver betöltésénél.
5. A start és end időpontok modulusát veszem az adapter által tárolt maximális időértékkel, és beírom őket a megfelelő TRGTTIM regiszterekbe.

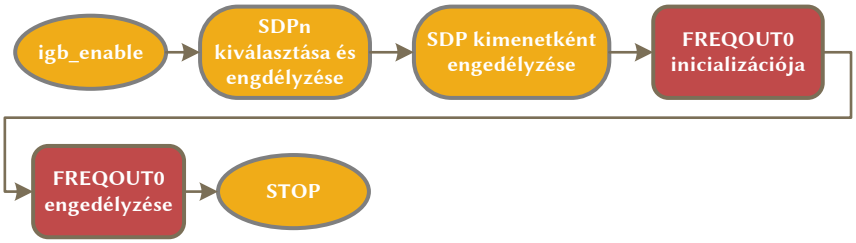
A *kiindulási időpontot eltárolom* az adaptert leíró struktúrában, a későbbi gyors felhasználhatóság végett, valamint beírom a target time regiszterek értékét, ezzel lezárva a kritikus szakaszt. Ebből az információból már egyszerű az újrainicializáció során továbbléptetni a további PPS élekhez tartozó target értékeket. Miután ezt elvégeztem, engedélyezem a target regiszterek használatát az `E1000_TSAUXC` regiszterben. Végezetül buszművelettel beírom a változtatásokat a regiszterekbe, kimaszkolom a megfelelő megszakításokat (`E1000_IMS`, `E1000_TSIM`

regiszterek), valamint inicializálok két kernel taskletet[2]. Ezek a rövid programok az interruptok lefutásakor kerülnek aktiválásra. Az egyik újraélesít a kimenetet, a másik pedig egy PPS eventet küld a kernelnek más órák szinkronizációjához. A taskletté kiszervezésük azért volt szükséges, mert az interruptkezelő hosszú futási ideje rontja a rendszer teljesítményét. A PPS kimenet újrainicializációját 800 ms-on belül meg kell tenni, a szoftveres PPS trigger is ugyanennyi belül élesítésre kell, hogy kerüljön, különben a továbbított timestringek nem a megfelelő PPS élekhez fognak tartozni!

A PPS kimenet leállításánál letiltom a target timerek működését, és kimaszkolom a hozzájuk kapcsolódó megszakításokat, valamint törlöm a végrehajtandó taskletek közül az inicializáláskor megadottakat.

12. ábra - PPS IT handler

Az újrainicializáció a PPS jel IT handlerében meghívott taskletekben történik. A PPS kimenet taskletje egy másodperccel növeli a `TRGTTIMx[01]` regiszter csoport értékét, és újraengedélyezi a működést az `E1000_TSAUXC` regiszterben. A PPS event taskletje egy `ptp_clock_event` típusú struktúrát tölti fel az aktuális PPS event adataival, majd pedig az eseményt regisztrálja a PPS alrendszerben.



13. ábra - Periodikus kimenet engedélyezése

A nagyfrekvenciás periodikus kimenet engedélyezésére a drivernek egy ptp_clock_request struktúrában átadjuk a kívánt SDP kimenet számát, és a periódusidőt. Az inicializáció hasonló az 1PPS jeléhez, az eltérés lényegében annyi, hogy a FREQOUT regiszter 8-2040 értékig fogad bemenetet, ami 16-4080 ns periódusidejű 50%-os kitöltési tényezőjű négyszögjel előállítását teszi lehetővé (245 kHz – 62,5 MHz). Ha kimenő jel fázishelyességének biztosításához szükséges, hogy a kimenet a következő másodperc felfutó élénél kerüljön aktiválásra, így az aktiválás rövid holtidővel jár.

mitpc37 – PTP slave eszköz, terhelésgenerátor

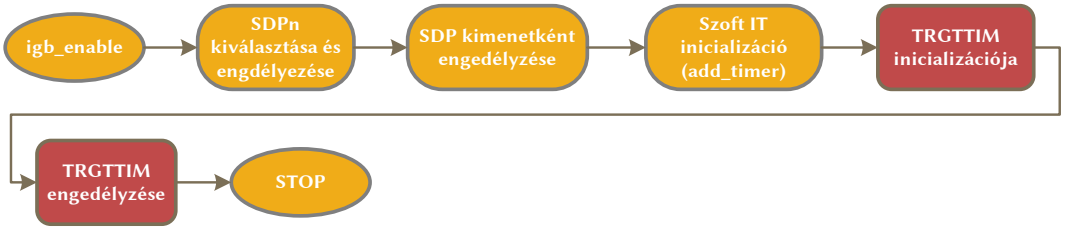
Ez a végpont kettős cél szolgál, egyrészt ezen is demonstrálom a PTP szoftver stack, és hálózati meghajtó működőképességét, másrészt műterhelésként funkcionál a hálózat számára. A számítógép egy Intel Pentium E2160-as processzort tartalmaz, és 2 gigabájt RAM-mal van

[2] A tasklet a meghajtóban ütemezett olyan feladat, amelynek a futását nem szükséges egy esemény bekövetkeztekor azonnal elindítani.

felszerelve. Az operációs rendszer az Ubuntu Linux 12.04, a használt SMP kernel verziószáma 3.2.0-ptp, azaz ez is egy egyedi kernel, mely modulként lefordítva tartalmazza a PHC API-t. A rendszer tartalmaz egy Intel 82580-as négy portos fejlesztői kártyát, melyek közül az egyik port csatlakoztatva van a lokális hálózatra a 192.168.1.2-es IP címmel. A 82580 óraszinkronizáció szempontjából megegyezik az i350-es kártyával, egy nagy megszakításkezeléssel kapcsolatos bugot leszámítva. A kártyán található kimenetek TTL jelszintűek. Ezt a kártyát Linux alatt szintén az 'igb' driver hajtja meg (v. 4.0.17). A rendszernek van menedzsment portja is, ez közvetlenül a tanszéki hálózatra kapcsolódik.

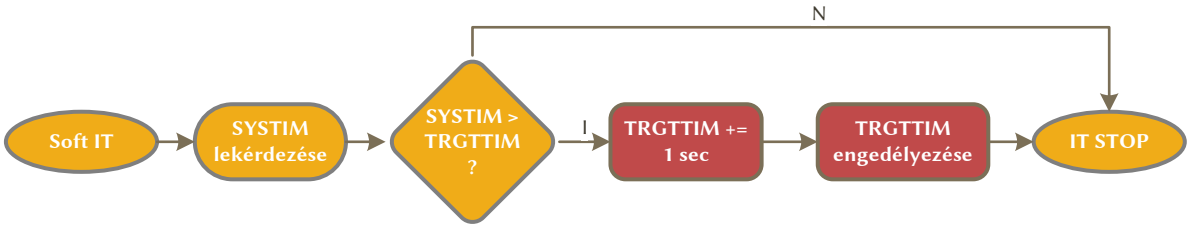
A PTP daemonok közül a ptpd2 és linuxptp daemonok működését itt is kipróbáltam, mindkettő segítségével sikerült a hálózati kártya hardveres óráját szinkronizálnom, megfigyeléseim megegyeznek az Ubuntu 11.10 VM részben leírtakkal.

Az 1PPS kimenet biztosítása a hardver hiányosságai miatt kizárólag hibrid módon lehetséges [7]. Ez azt jelenti, hogy habár lehetséges a kimenetekre az előzőleg tárgyalt belső clock trigger üzemódok alapján impulzusokat generálni, az ehhez tartozó megszakítások nem kiolvashatóak, ezért a kimenet újrainicializációja a megszakításkezelőből nem lehetséges, csak úgy, hogy periodikusan olvassuk a belső órát, és ha túlhaladtuk azt, akkor újrainicializáljuk a kimenetet.



14. ábra - hardveres kimenet inicializációja hibrid módon

Ez annyiban tér el a más ismertetett módszertől, hogy a megszakítások engedélyezése helyett egy szoftveres időzítőt kell létrehoznunk, mely segítségével 250 ms-ként lekérdezzük a kártya órájának értékét, és amennyiben az nagyobb, mint a legutóbbi lefutó élre beprogramozott idő, újrainicializáljuk a kimenetet.



15. ábra - hardveres kimenet újrainicializációja hibrid módon

Az újrainicializáció már egy egyszerűbb eset – lekérdezem, hogy a kártyában mennyi idő telt el a legutóbbi polling óta, ha ez kevesebb, mint hogy nekünk újrainicializálni kellene, akkor kilépek a függvényből a következő periodikus újrahívásig. Amennyiben újra kell élesítenem a kimenetet, akkor megnövelem a target regiszterek értékeit egy másodperccel, engedélyezem a működésüket, majd kilépek a rutinból.

Ennek a megoldásnak a nagy hátránya az, hogy az alkalmazott nem realtime Linuxban semmilyen garancia nincs arra, hogy ezek a szoftveresen időzített rutinok az általunk megadott periódusidő szerint fognak elindulni. Természetesen mivel a rendszer számítási kapacitása aránylag nagy, és több száz milliszekundum periódusidejű lekérdezésekhez ms nagyságrendű ingadozások is elfogadhatóak az újrahívás periódusidejében, ezért ez a megoldás nagy valószínűséggel sok esetben alkalmazható.

PicoScope 5203 – független mérési állomás

Az általam használt PicoScope egy hordozható oszcilloszkóp, melynek az adatait egy számítógépen futó program segítségével lehetséges kiértékelni. A szkópnak két bemenő csatornája van, valamint képes fogadni külső trigger jelet, és van benne hullámforma-generátor is. A mért adatokat USB-n keresztül juttatja a számítógépbe.
A mintavételezési sebessége 1 GS/s egyetlen csatornán, két csatornán ez megfeleződik. Mivel én a méréseimhez két csatornát használtam, ezért effektíve 2 ns felbontással tudtam megfigyelni a jelalakokat, ez elegendő felbontás a mérés kiértékeléséhez.

Mérési eredmények

Terhelésgenerálás iperf segítségével

Ahhoz, hogy hálózati terhelés alatt is tesztelhessem a rendszert, az iperf alkalmazást telepítettem a PTP slave gépekre. A mérési eredmények kiértékeléséhez fontos, hogy a műterhelés működését kifejtsem. Az iperf egy kliens-szerver architektúrájú eszköz, amely képes forgalom generátorként (kliens) és nyelőként (szerver) üzemelni. A szervert az

```
$ iperf –s –B <IP cím> -u
```

paranccsal indítva egy UDP socketet nyitunk az <IP cím> IP-jű hálózati csatolón. A klienst az

```
$ iperf –c <szerver IP> -u –t <T> -b <sávszélesség>
```

paranccsal indítva csatlakoztathatjuk a szerverre, és T ideig generálhatjuk a forgalmat, az általunk beállított sávszélességgel. A szerver az érkező csomagokat automatikusan eldobja, feldolgozást rajtuk nem végez. Az –u kapcsoló használata nélkül TCP kapcsolatot építhetünk ki az állomások közt, ekkor a –b kapcsolónak nincs értelme, hiszen a TCP mindig kihasználja a rendelkezésre álló sávszélességet.

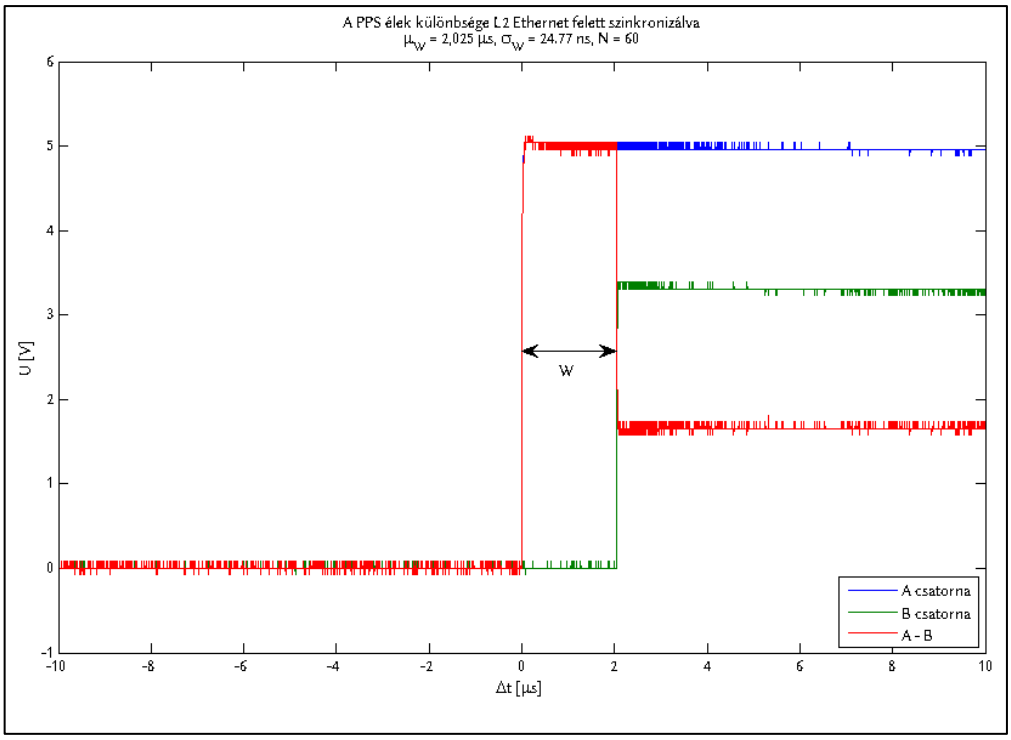
A mérési eredmények kiértékeléséhez fontos tudni, hogy az 1000BASE-T Ethernet full-duplex protokoll, azaz különálló adó és vételi csatornája van, melyeken egyidőben képes kommunikálni. Teljesen természetes jelenség az, hogy a kliens slave gépen a beérkező szinkronizációs üzenetek nagyobb késleltetés nélkül megérkeznek, és csak a terjedési idő számításánál akadnak kisebb-nagyobb hibák, míg a szerver szerepben a szinkronizáció pontossága romlik, hiszen a beérkező csomagok feltorlódnak a switchben (amennyiben a switch nem transparent clockként viselkedik).

Mérési beállítások

A méréseket elvégeztem mind UDP, mind Ethernet felett szinkronizálva. Az oszcilloszkóp beállítása úgy történt, hogy az A csatornára vezetett master PPS referenciajel felfutó élére történt a mintavételezés, a B csatornán pedig a slave eszköz PPS jele volt látható. Mivel a periódusidő 1 másodperc volt, ezért másodpercenként új regisztrátum született, időalaptól függően 5000-500000 mintából. Az oszcilloszkóp szoftverben beállíthatóak matematikai műveletek a csatornákkal, én a kivonás műveletét állítottam be, és a létrejövő pszeudo-impulzus szélességét vizsgálva mértem meg az ofszetet, és szórást. Erre az oszcilloszkóp beépített mérési funkciókat tartalmaz, beállíthatjuk a mérési minták számát, kijelzi a maximális és minimális értékeket, az átlagot, és a szórást. Mivel a PPS generátor kimenete TTL szintű, viszont a hálózati kártya bemenetei csak 4 V bemenő feszültségig fogadnak jelet, ezért az erre történő lokális esemény időbélyegzés nem valósult meg, így idődiagramot sem tudtam készíteni a szinkronizáció folyamatából.

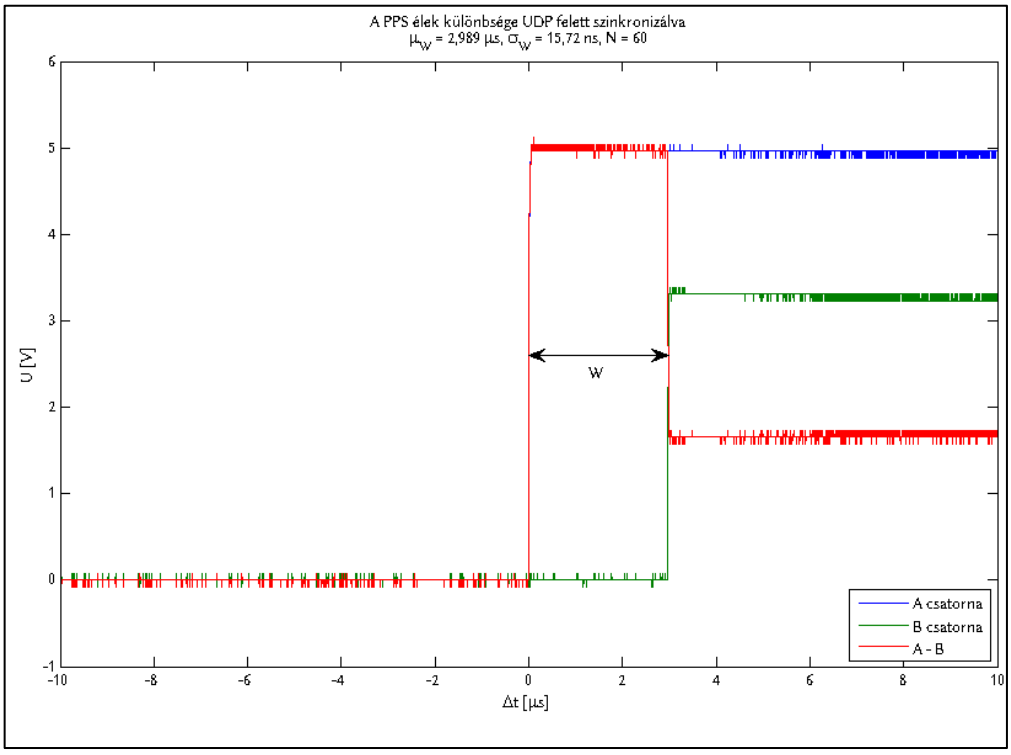
Szinkronizáció első körben

Először az Ethernet feletti szinkronizációt teszteltem a master óra segítségével:



Az eredményből látható, hogy a két jel felfutó éle közti időkülönbség a célon felüli, viszont az élek távolsága közti szórás nagyon alacsony.

A következő ábrán az UDP feletti szinkronizációt láthatjuk:



Itt is jól kivehető, hogy a konstans eltérés van a két él között, kis szórással. Mivel az eltolás átlaga a szóráshoz képest két nagyságrenddel nagyobb, ezért lehetséges volt ezt konstans hibatagként kezelni a driverben. Mivel a különböző szállítási rétegek közt eltérő volt az eltérés nagysága, ezért azt az igb driver betöltésekor lehet megadni.

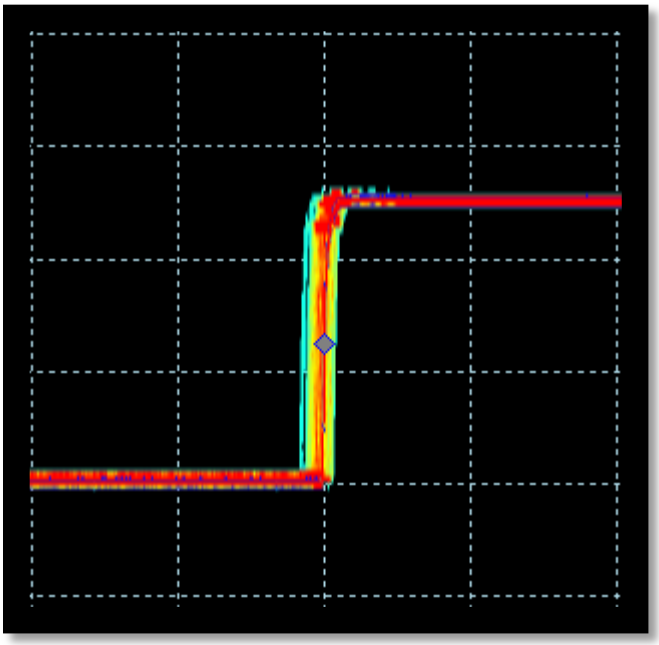
```
$ modprobe igb PPS_COMP=[nsec]
```

A paraméter értéke a kívánt korrekció idejét adja meg. Mivel egyetlen tesztrendszer állt rendelkezésre, ezért a késleltetéseket befolyásoló tényezőket nem sikerült minden részletre kiterjedően megmérni. Lehetséges, hogy különböző rendszerekben eltérőek ezek az értékek, ezért nem került behuzalozva a driverbe. A kimért késleltetések kompenzációja az iparban elfogadott módszer, azonban fontos a továbbiakban is foglalkozni vele, mert lehetséges, hogy a szabályozási kör egy hibájára hívja fel a figyelmet. A driver betöltéskor kötött korrekció nem jelent komoly megkötést egy valós rendszerben, ugyanis ekkor már nem szokás változtatni a szállító közeg típusát. A korrekció mértékét egy éles rendszerben úgy választanám meg, hogy a szórás kétszerével kisebbek legyenek, mint az átlagértékek, ugyanis így a slave eszköz felfutó éle a master eszközé után következik az esetek túlnyomó többségében, ezáltal nem sérülnek bizonyos kauzalitási feltételek például egy vezérlési hálózatban. A teszteseteket úgy futtattam le, hogy akár pozitív irányban is növeltem a terhelés nélküli eltérést azért, hogy a szkóp szoftverével a statisztikákat ki tudjam értékelni.

Szemábrák

Szemábrának nevezzük a telekommunikációban azt a diagramot, amely úgy jön létre, hogy a jelfolyamban átküldött szimbólumokat ábrázoljuk az oszcilloszkópon a digitális foszfor funkció segítségével. Az általam mutatott szemábrák kizárólag 0-1 átmenetet ábrázolnak, hiszen az 1PPS jel információtartalmát ennek az élnek a helyzete tartalmazza.

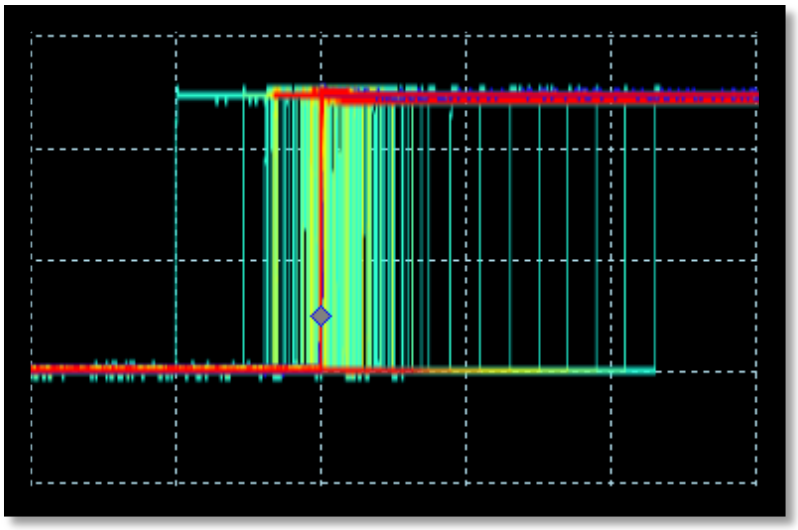
A korrekció alkalmazása után (2900 ns) hálózati terhelés nélküli esetben az alábbi szemábra jött létre:



16. ábra – a slave kimenetének szemábrája 30 perc folytonos szinkronizáció után (dX = 0,5 us, dY = 2 V)

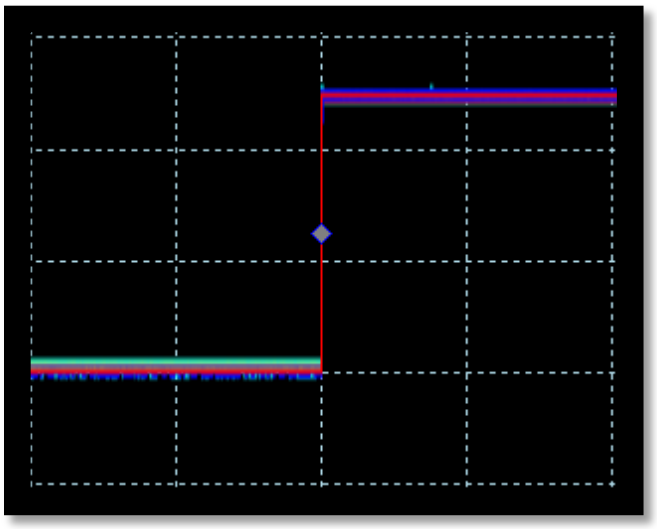
Az előbbi ábrán a függőleges tengely mentén a slave eszköz jele 1,48x-osára volt skálázva, így látszólagos amplitúdója megegyezett a master eszközével, teljesen fedte azt (3,3V-ról 5V-ra). Az ábrán a trigger helyét a szürke csúcsára állított négyzet mutatja.

Amennyiben a hálózaton műterhelést szimulálunk, a szinkronizálódó slave eszköz pontossága változni fog, hiszen a továbbított csomagok a switchben véletlenszerűen feltorlódnak, és a Transparent Clock működést nem támogató eszköz ezt a véletlenszerű késleltetést nem kompenzálja a csomagokban. Az alábbi ábrán a kliens 1PPS jelét láthatjuk az iperf szerver futtatása esetén. Ekkor a kimenet átlagos eltérése 1,03 us, szórása 3,18 us volt.



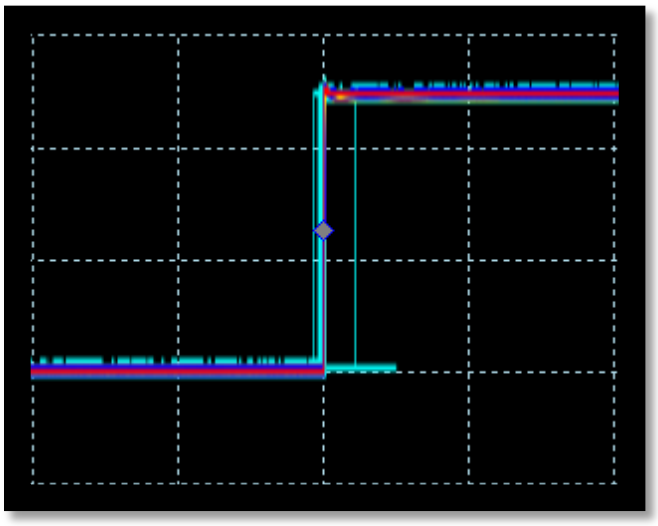
17. ábra – a slave PPS kimenete 5 perc 500 Mb/s bemenő forgalom esetén (dX = 10 us, dY = 2 V)

A iperf kliens elindítása esetén a kimenet a terhelés nélkül képet mutatta (16. ábra), hiszen a bejövő forgalomban nem torlódtak fel a SYNC üzenetek, így a switchben a késleltetés-ingadozások ebben az irányban nem okoztak mérési hibát. Előfordult, hogy a késve érkező DELAY_REQUEST/DELAY_RESPONSE csomagok miatt a szabályozási kör hibás munkapontba állt be, a rendszer kilendült egyensúlyi állapotából, és ezért a szinkronizáció pontossága elmaradt az elvárt értéktől. Ennek kiküszöbölése csak a forgalom priorizálásával, vagy Transparent Clock alkalmazásával lehetséges.



18. ábra – a kimenet stabilitása hosszú távon (t = 24h, dX = 20 ms, dY = 2 V)

Az implementáció jelenlegi állásában előfordulnak apróbb hibák, habár hosszú távon a kimenet stabil (18. ábra), néha akadnak benne hibás jelformák egy-egy periódus erejéig (19. ábra). Ezek alkalmazástól függően elfogadhatóak, vagy sem, kiküszöbölésük további méréseket igényel. A 19. ábra által mutatott glitch a kijelzésben (az alacsony jelszint túllóg a megelőző élen) az oszcilloszkóp programjának tökéletlensége miatt jelenik meg, ott valójában még egy darab él található, azonban ez az exportból konzisztensen eltűnt.



19. ábra – a kimenet stabilitása hosszú távon (t = 24h, dX = 10 us, dY = 2V)

Összehasonlítás a korábbi eredményekkel

Az összehasonlíthatóság a korábban mért eredményekkel lehetséges, azokban a korábbi publikációkhoz [13] [14] képest az iperf szerver szerepben mért eredmények szolgálnak újdonsággal. A szabályozási kört megvalósító szoftver is változott, a ptpd/ptpd2 le lett cserélve, a linuxptp szoftveres óraállítási mechanizmusát nem teszteltem.

2. táblázat – a mérési eredmények összehasonlítása korábbi munkákkal

Szinkronizálási eljárás	Mérési szituáció	Ofszet átlaga	Ofszet szórása
PTP szoftveres megvalósítással	Terhelve	53 μs	46 μs
	Terheletlenül	3,55 μs	3,8 μs
PTP hardver támogatva (kernel óra állítás)	Terhelve	750 ns	220 ns
	iperf kliens	603 ns	133 ns
PTP hardver támogatva (hardver óra állítás)	Terheletlenül	13 ns	24,77 ns
	iperf szerver	1,03 μs	3,18 μs
	iperf kliens	22 nsμ	27,41 ns

Az eredményekből látható, hogy a terheletlen esetben az átlag és a szórás egyaránt egy nagyságrendet csökkent, nem beszélve arról a tényről, hogy immáron valóban független eszközzel, a számítógépen kívülről vizsgálható a szinkronizáció pontossága. Az ennél nagyobb pontosság eléréséhez már olyan rendszerkomponensek szükségesek, melyek felbontása a nanoszekundum alatti tartományokban van, és komoly rezgés és hőmérsékletkompenzációval rendelkezik.

Hordozhatóság beágyazott rendszerekbe

Az általam bemutatott megoldás egy nagy előnye, hogy megfelelő hardveres támogatás esetén egyszerűen hordozhatjuk azt beágyazott környezetbe. A fő megkötéseink egyike a Linux kernel futtathatósága az adott rendszeren – ezzel nagyjából a nagyobb 32 bites mikrovezérlőkre korlátoztuk a rendszert vezérlő processzorokat – a másik a rendszerben található kontrollálható hardverben implementált óra megléte. Ha ez a két feltétel teljesül, akkor a megfelelő driverek jelenlétében az általam javasolt megoldás egy- az-egyben átültethető rájuk, akár nem valós idejű Linuxot alkalmazva is.

Ha olyan környezetben kívánjuk a rendszert működtetni, ahol a futtató környezet valós idejű Linux, akkor az autonóm időbélyegzés használata nélkül, hardveres megszakítások használatával csökkenő, de alkalmazástól függően elégséges pontosság mellett vagyunk képesek a szinkronizációra. Az IA-32, AMD64 alapú személyi számítógépes architektúráktól eltérően a beágyazott rendszerekben a hardverben történő időkezelés általánosan támogatott, ezért gyakran a meglevő eszközökön is lehetséges kielégítő eredménnyel implementálni az algoritmust. Fontos tehát, hogy szétválasszuk a hardveres órák állításának problematikáját (melyre a PHC API az esetek túlnyomó többségében megoldást tud nyújtani) a hálózati időbélyegek létrehozásának problémájától. Nagy pontosságot igénylő alkalmazásokban a hardveresen támogatott időbélyegzés megléte elengedhetetlen, azt rendszerszintű megszakítások által pótolni nem lehetséges.

Kisebb teljesítményű beágyazott rendszerekben is létezik PTP megvalósítás, így az interoperabilitás biztosítása is fontos. Amennyiben a szoftver mindent a szabvány szerint valósít meg, akkor csak konfigurációs problémák jelenhetnek meg például a végeszközök E2E/P2P beállításai közt, melyek kiküszöbölésére fokozottan oda kell figyelnünk.

Összefoglalás, továbblépési lehetőségek

A TDK dolgozatban prezentált rendszert egy rendszerórától független külső megfigyelő segítségével kisméretű teszthálózatban ellenőriztem, demonstráltam a megoldás működőképességét, azonban még további mérések elvégzése szükséges. A továbbiakban több kézenfekvő továbblépési lehetőség kínálkozik.

A validáció idődiagramjainak előállításhoz egy szintillesztő kapcsolás tervezése szükséges, mivel a referenciaóra 1PPS kimenetének jelszintje közvetlenül nem illeszthető a kártya PPS bemenetére. A kapcsolás karakterisztikáját precízen kell bemérni, mivel a hagyományos olcsó ellenállásokból épített feszültségosztók által tartalmazott parazita kapacitások, induktivitások a kimért 1PPS jel szórásával megegyező tartományba esnek. A kártya külső esemény időbélyegzésével tudja megállapítani az eltérést az órája és referencia közt. Az idődiagramok előállításával megfigyelhetnénk, hogy a szinkronizációs lánc mely pontján keletkezi az 1PPS kimenetek késleltetése.

A rendszert körülvevő valós kommunikációs hálózatba illesztése során érdemes megvizsgálni az olyan eseteket, amikor nem áll rendelkezésre PTP funkcióira felkészített hálózati infrastruktúra (például Transparent Clockok, Boundary Clockok). Egy mérés témája lehet a meglevő infrastruktúra 802.1p Qos szolgáltatásának bekonfigurálása, úgy hogy a PTP csomagokat priorizált forgalomként kezelje. Ekkor elvileg switchenként a maximális késleltetés mértéke egyetlen csomag adási ideje lenne, hiszen a hagyományos Ethernet nem képes preemptcióra, egy elkezdett keret adását nem lehet megszakítani.

Érdemes lehet megvizsgálni egy valós infrastruktúrában elhelyezett Boundary Clockok, Transparent Clockok hatását. Ehhez jelenleg a tanszéken rendelkezésre állnak TTEthernet switchek, melyek a L2 Ethernet felett képesek Transparent Clock szerepet betölteni, azonban csak optikai portokkal rendelkeznek, az általam bemutatott rendszer jelenlegi kiépítése pedig réz hordozóra épül. Szerencsére a szerver adapterek léteznek optikával felszerelt kivitelben is, így mérésekre alkalmas a hálózat. Ezen a hálózaton természetesen valós beágyazott környezetbeli méréseket nem lehet elvégezni, hiszen ilyen közegben ritkább az optikai interfész használata.

Érdekes probléma a Boundary Clock funkció implementációja többportos kártyák esetén. Az i350 hardvere szükségessé teszi, hogy az adapterek belső óráit is szinkronizáljuk egymással. A linuxptp segítségével a szoftveres megvalósítás elő van készítve.

Az egyik legfontosabb rövid távú cél a 6. ábra ábrán jelölt PTP master és referenciaforrás szerepek saját kézzel történő integrálása egyetlen eszközbe, így teljesedne ki a rendszer egy ellenőrizhető teljesen nyílt forráskódú szoftveren alapuló megoldássá. Itt a fő feladat a GPS adó timestring és PPS jelének feldolgozása a PPS stack és a hálózati kártya SDP bemenetei által.

Ábrajegyzék

Irodalomjegyzék

[1] IEEE Instrumentation and Measurement Society Std., IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, 2008.

[2] J. Eidson, M. Fischer és J. White, „IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control systems," in *34th Annual Precise Time and Time Interval (PTTI) Meeting*, 2002.

[3] Industrial Ethernet Book, „IEEE 1588v2: Precision timing for substation automation," 2010. [Online]. Available: http://-www.iebmedia.com/-index.php?id=7048&parentid=74&themeid=255&hpid=2&showdetail=true&bb=1&appsw=1&sstr=1588 . [Hozzáférés dátuma: 11 10 2012].

[4] R. Boatright, „Understanding IEEE's new audio video bridging standards," EE Times, 2009. [Online]. Available: http://www.eetimes.com/design/audio-design/4008284/-Understanding-IEEE-s-new-audio-video-bridging-standards?pageNumber=1 . [Hozzáférés dátuma: 08 09 2012].

[5] Symmetricom, „Cesium Atomic Clock - Cesium Beam Frequency and Time Standard Instruments," [Online]. Available: http://www.symmetricom.com/products/frequency-references/cesium-frequency-standard/. [Hozzáférés dátuma: 01 10 2012].

[6] Intel, „Hardware-Assisted IEEE 1588* Implementation in the Intel® IXP46X Product Line alapján," Intel, [Online]. Available: http://www.intel.com/design/network/products/npfamily/ixp460.htm. [Hozzáférés dátuma: 13 10 2012].

[7] C. Richard és C. Marinescu, „Design and implementation of a PTP clock infrastructure for the Linux kernel," in *Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, Portsmouth, NH, 2010.

[8] Meinberg Funkuhren, 2010. [Online]. Available: http://www.meinberg.de/download/docs/manuals/english/m600_ptp2.pdf. [Hozzáférés dátuma: 20 10 2012].

[9] Intel, „Intel® Technology Journal," [Online]. Available: http://www.intel.com/technology/itj/2006/v10i3/2-io/1-abstract.htm. [Hozzáférés dátuma: 03 10 2012].

[10] P. v. d. Does, „Compile Linux kernel 3.2 for Ubuntu 11.10 (Oneiric Ocelot)," 2012. [Online]. Available: http://blog.avirtualhome.com/compile-linux-kernel-3-2-for-ubuntu-11-10/. [Hozzáférés dátuma: 30 09 2012].

[11] Intel, „i350 Datasheet," 2011. [Online]. Available: http://www.intel.com/content/www/us/en/ethernet-controllers/ethernet-controller-i350-datasheet.html. [Hozzáférés dátuma: 13 10 2012].

[12] Intel, „Intel 82580 datasheet," 2010. [Online]. Available: http://www.intel.com/products/-ethernet/resource.htm#s1=Gigabit%20Ethernet&s2=82580EB/82580DB&s3=Datasheet. [Hozzáférés dátuma: 11 09 2012].

[13] T. Kovácsházy és B. Ferencz, „Hardware Assisted IEEE 1588 Clock Synchronization for Linux Based Network Embedded Systems," *CARPATHIAN JOURNAL of ELECTRONIC and*

COMPUTER ENGINEERING, p. 11, 2011.

[14] T. Kovácsházy és B. Ferencz, „Performance evaluation of PTPd, a IEEE 1588 implementation, on the x86 Linux platform for typical application scenarios," *Proceedings of the 2012 IEEE International Instrumentation and Measurement Technology Conference (I2MTC 2012),* pp. 2548-2552, 2012.

[15] D. Mills, „Internet time synchronization: The network time protocol," *Communications,* %1. kötet39, pp. 1482-1493, 1991.