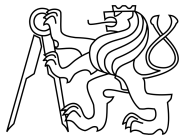
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Monitor síťového provozu v protokolu PTP
Student:	Vlastimil Lhotecký
Vedoucí:	RNDr. Ing. Vladimír Smotlacha, Ph.D.
Studijní program:	Informatika
Studijní obor:	Informační technologie
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Navrhněte a naprogramujte v Linuxu aplikaci pro monitorování provozu časového serveru s protokolem IEEE1588 / PTP. Pro zachycení vybraných paketů použijte knihovnu PCAP. Aplikace poskytuje v reálném čase seznam identifikovaných serverů a klientů, parametry komunikace mezi nimi a stav hodin jednotlivých účastníků. Aplikace bude pracovat ve dvou režimech: pasivním, kdy sleduje komunikaci jiných účastníků, a aktivním, kdy se sama stane klientem s využitím existující SW implementace protokolu IEEE1588 / PTP. Výstup aplikace bude textový bez grafického uživatelského rozhraní.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 10. ledna 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

Monitor síťového provozu v protokolu PTP

Vlastimil Lhotecký

Vedoucí práce: RNDr. Ing. Vladimír Smotlacha, Ph.D.

30. června 2017

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu mé bakalářské práce RNDr. Ing. Vladimíru Smotlachovi, Ph.D. za odborné vedení, trpělivost a cenné rady, které mi v průběhu zpracování bakalářské práce poskytl.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 30. června 2017 .....

Odkaz na tuto práci

Lhotecký, Vlastimil. *Monitor síťového provozu v protokolu PTP*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

V této práci se zabývám návrhem a implementací programu pro monitorování síťového provozu v protokolu PTP v rámci lokální sítě v reálném čase. Důležitou informací je hlavně přesnost synchronizace času mezi monitorovanými stroji a časovým serverem a případné chyby v komunikaci.

Během práce jsem provedl analýzu protokolu a zhodnotil, zda by pro daný účel nestačilo použít některý z univerzálních nástrojů pro monitorování sítí, jako je např. Wireshark. Dále jsem navrhl vlastní nástroj a ten následně implementoval v jazyce C++ s použitím knihovny PCap a OS Linux.

Klíčová slova počítačové sítě, synchronizace času, Precision Time Protocol, PTP, monitoring, hledání závad, Linux

Abstract

The aim of this thesis was to design and create a network monitoring tool, that would capture and report network traffic via the PTP protocol in real time, with emphasis on precision of the distributed time as well as possible errors in communication.

Firstly, I go over the fundamentals of PTP and try to determine whether a universal monitoring tool, such as Wireshark would be suitable for this

purpose. In the second part of this document I design my own monitoring tool for OS Linux and subsequently implement it using C++ programming language with the PCap library.

Keywords computer networks, time synchronization, Precision Time Protocol, PTP, monitoring, troubleshooting, Linux

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Co je PTP a kde se používá

PTP (standard IEEE 1588) je zkratka z anglického Precision Time Protocol, volně přeloženo jako Protokol přesného času. Jak název napovídá, jedná se o síťový protokol sloužící k synchronizaci času s důrazem na přesnost. PTP podle specifikace dosahuje přesnosti v řádu nanosekund (nejvyšší rozdíl času mezi dvěma zařízeními). Protokol je možné provozovat nad mnoha různými technologiemi, dvě nejrozšířenější možnosti jsou přenos PTP zpráv uvnitř UDP datagramu (čtvrtá vrstva OSI) a přenos PTP zpráv přímo uvnitř Ethernetového rámce (druhá vrstva OSI).

Přestože je PTP běžný aplikačný protokol, pro který existují softwarové implementace serveru i klienta, v běžných sítích se většinou nepoužívá. Jednak není tak vysoká přesnost např. pro firemní LAN potřeba, navíc je pro její dosažení potřeba hardwarová podpora na všech síťových prvcích mezi klientem a serverem. Bez této podpory je přesnost protokolu přibližně stejná jako u NTP, jehož klient bývá implicitně obsažen ve všech běžně používaných operačních systémech.

Typickou oblastí použití PTP jsou měřící a řídící aplikace v oblasti výzkumu nebo průmyslu, kde se nejčastěji v roli serveru používá dedikované zařízení, které jako zdroj přesného času využívá signál GPS nebo atomové hodiny. [1]

motivace a cíle

Při nasazení protokolu PTP v síti je často potřeba ověřit, zda distribuovaný čas na jednotlivých klientech skutečně odpovídá času serveru s příslušnou tolerancí. Bohužel samotný protokol tuto funkcionalitu nezahrnuje a tak je nutné porovnávání provádět ručně. Cílem práce je navrhnout a implementovat nástroj, který bude v reálném čase zobrazovat čas distribuovaný protokolem

PTP na všech klientech v lokální síti a porovnávat ho s referencí. Tím bude umožněno monitorovat funkci PTP v dané síti. Informace o klientech budou získávány odposlechem jejich komunikace na síti.

Protože informace získané tímto způsobem mají omezenou přesnost, bude mít nástroj kromě režimu monitoru ještě režim klienta, ve kterém bude zobrazovat přesnější parametry komunikace, ale pouze pro to zařízení, na kterém poběží.

Kapitola 1

Principy protokolu PTP

1.1 Popis protokolu

Precision Time Protocol byl poprvé standardizován v roce 2002 jako IEEE 1588 - síťový protokol pro synchronizaci času v distibuovaných měřících a kontrolních systémech - a určen pro synchronizaci hodin nezávislých zařízení zapojených v síti s vysokou přesností. [2]. Přesnost synchronizace je specifikována v řádu nanosekund. V roce 2008 byla standardizována druhá revize protokolu, která není zpětně kompatibilní a kromě změny formátu zpráv zavádí nový druh hodin - Transparent Clock - a alternativní model synchronizace. Protokol se stále vyvíjí, připravuje se verze 3, která by měla řešit některé z problémů předchozích verzí. [3] I když je protokol specifikován jako značně univerzální, jeho nasazení je vhodné spíše v jednodušších sítích, s rostoucím rozsahem sítě se zvyšuje počet faktorů, které negativně ovlivňují přesnost synchronizace. [2]

Protokol je nenáročný na výkon zařízení a šířku pásma, je ho tedy možné kromě PC nasadit i pro synchrinizaci různých vestavných systémů využívajících jednodušší mikrokontrolery.[4] Je ho také možné provozovat nad různými linkovými protokoly, pro účely této práce budu jako linkový protokol předpokládat Ethernet. PTP zprávy mohou být přenášeny přímo v ethernetovém rámci, nebo zabaleny do IPv4, případně IPv6 UDP packetu. V takovém případě se implicitně využívá multicast komunikace, ale je možné nastavit i unicast, který se používá především v oblasti telekomunikací. [5]

1.1.1 Základní Pojmy

PTP zavádí několik pojmů, jako *Clock (hodiny)* je chápáno jakékoli zařízení, které je schopno s definovanou přesností udržet nastavený čas. Rozšířením tohoto pojmu je pak jakékoli zařízení v síti schopné PTP komunikace. Protokol definuje tři typy hodin, *Ordinary Clock*, *Boundary Clock* a *Transparent Clock*, jejich popis je v sekci 1.2. Každé hodiny mohou být buď v režimu *Master*,

kdy slouží jako zdroj času pro ostatní zařízení v síti nebo *Slave*, kdy si čas nastavují podle předřazeného mastera. Master hodiny s nejpřesnějším časem se nazývají *Grandmaster*. Údaj o čase je vyjádřen jako *Timestamp (časová značka)*, což je číslo udávající uplynulý čas od nějakého okamžiku (epochy). Přesný formát časových značek je v sekci 2.4

1.2 Druhy zařízení

Zařízení v síti lze rozdělit následujícím způsobem. V této sekci se pojem *port* chápe jako síťové rozhraní v PTP topologii.

1.2.1 Ordinary Clock (OC)

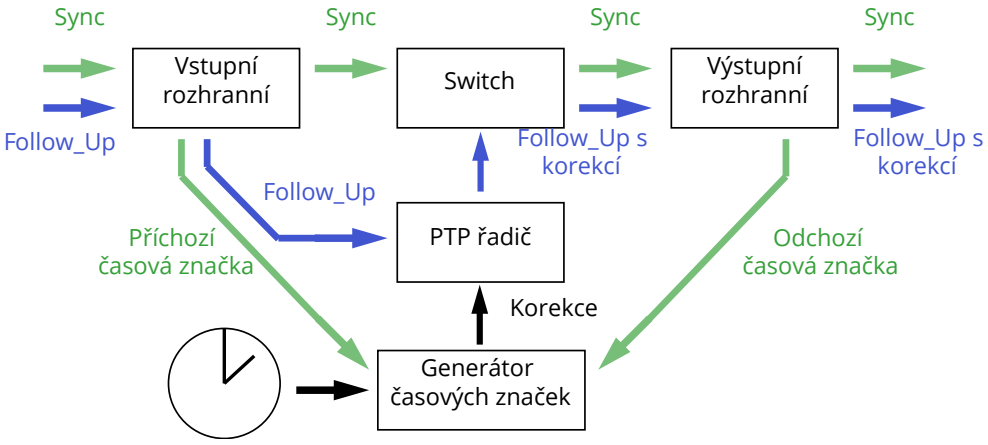
Jako Ordinary Clock (běžné hodiny) se označují PTP zařízení s jedním síťovým portem, který může být v režimu Master, Slave nebo Passive. Port v režimu Master v pravidelných intervalech vysílá synchronizační zprávy a zároveň odpovídá na dotazy o zpoždění přenosové cesty. Port v režimu Slave přijímá synchronizační zprávy podle kterých si zařízení nastavuje čas a dále pravidelně posílá dotazy na zpoždění přenosové cesty. Synchronizační zprávy a dotazy na zpoždění zajišťují dva nezávislé, paralelně běžící procesy.[6] Detaily synchronizace v sekci 1.3.2. Port v režimu Passive má zařízení, které z nějakého důvodu chce mít možnost PTP komunikace, ale nechce pomocí něj synchronizovat čas. Tento mód se využívá typicky pro záložní hodiny, které při výpadku Mastera převezmou jeho fukнci.

1.2.2 Boundary Clock (BC)

Boundary Clock (hraniční hodiny) jsou zařízení s více PTP porty. Jeden z portů se chová jako Slave, pomocí něj si BC sesynchronizuje vnitřní hodiny podle předřazeného mastera. tento čas následně distribuuje dál pomocí ostatních portů, které jsou v režimu Master. 1.3 Boundary Clock může představovat např. switch nebo router. Díky konceptu boundary clock se dá PTP nasadit i v rozsáhlé síti, každé dalsi BC po cestě od synchronizovaného zařízení ke Grandmasterovi ale snižují přesnost jeho synchronizace, protože se musí nejprve samy synchronizovat podle předřazeného Mastera a až potom mohou předávat informaci o čase dál. Tento problém částečně řeší E2E Transparent Clock.

1.2.3 Transparent Clock (TC)

Transparent Clock *(transparentní hodiny)* jsou asi nejzajímavější z PTP zařízení, které byly zavedeny v druhé verzi protokolu IEEE 1588-2008. V této sekci používám některé pojmy ze sekce zprávy **??** Podle použitého řežimu kominikace 1.3.3 se TC dělí na dva druhy



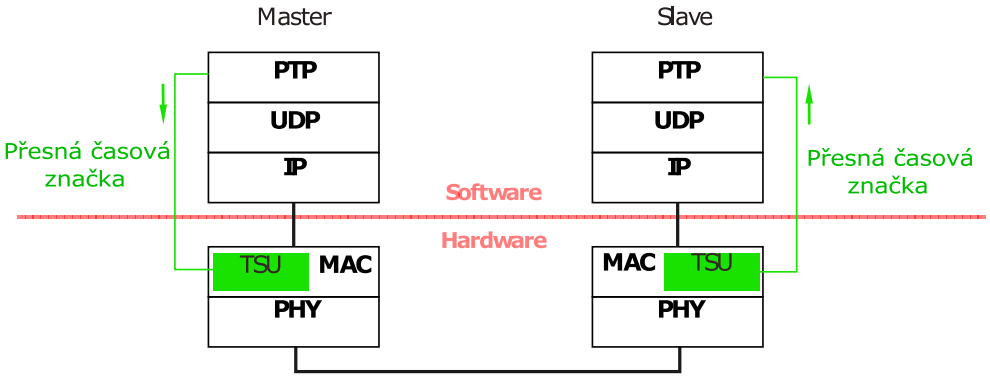
Obrázek 1.1: Činnost transparentních hodin v E2E režimu.

End-to-End Transparent Clock - V E2E režimu TC nezpracovávají PTP zprávy (proto transparentní), pouze provádí časovou korekci u zpráv, které jimi prochází. Tato korekce slouží ke kompenzaci zpoždění způsobeného průchodem packetu zařízením, kdy packet může relativně dlouhou dobu čekat ve frontě na zpracování. Pokud je přijatý packet PTP zpráva, TC si zaznamená časovou značku jejího přijetí a čeká na zpravování packetu. Na výstupu ze zařízení si TC opět zapíše časovou značku, a z rozdílu těchto dvou značek spočte dobu kterou packet strávil v zařízení. Toto zpoždění se zapíše do hlavičky odcházejícího PTP packetu. Tímto je zajištěno, že zpoždění způsobené čekáním packetu v zařízení se započítá do celkového zpoždění přenosové cesty. Pro zvýšení přesnosti může TC zapsat zpoždění procházející Sync zprávy do následující Follow_Up zprávy (v případě Two-Step synchronizace) 1.1 a zpoždění procházející Delay_Req zprávy do příslušné Delay_Resp zprávy (pokud není žádná možnost, že Delay_Resp půjde jinou cestou než Delay_Req). [6]

Peer-to-Peer Transparent Clock - Používají se při P2P způsobu měření zpoždění přenosové cesty, vice v sekci 1.3.3. Každé TC si pomocí P2P zpráv (Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up) nejdříve změří zpoždění přenosové cesty k dalšímu uzlu sítě a při průchodu Sync zprávy do ní toto zpoždění zapíše (připočte k současné hodnotě). Slave zařízení tedy při přijetí zprávy zná zpoždění celé přenosové cesty a může hned vypočítat posunutí svých hodin vůči Masterovi. [7]

1.2.4 Ostatní

Přesnost v řádu nanosekund je u PTP možná díky hardwarovým časovým značkám. K tomu je nutný síťový hardware, který je podporuje. U běžných síťových karet packet po přijetí čeká v bufferu než na něj přijde řada a ovla-



Obrázek 1.2: Princip hardwarových časových značek.

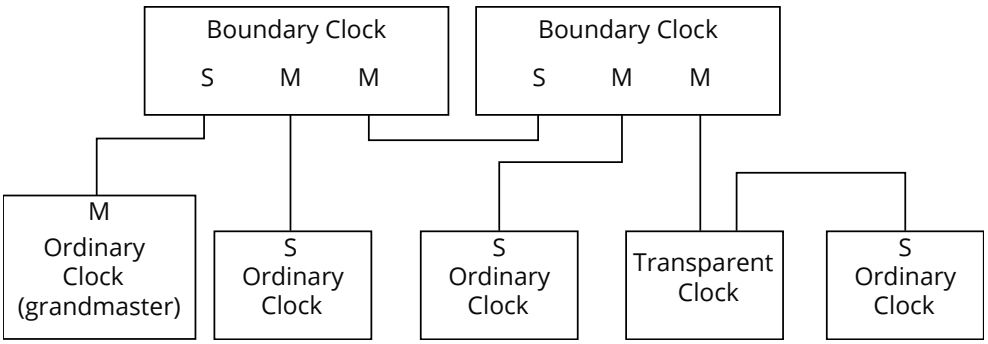
dač síťové karty z něj vybalí Ethernetový rámec, který předá jádru OS, kde se postupně rozbalí na IP packet, UDP datagram a konečně PTP zprávu. Toto probublání vrstvami způsobuje zpoždění mezi přijetím zprávy a jejím zpracováním. Pokud by zpoždení bylo konstantní, nebyl by to problém, to ale nelze zaručit a ve zpoždění jednotlivých zpráv mohou být značné rozdíly (stack jitter) a tím pádem se přesnost synchronizace zhorší. Podobná situace nastává u switchů a routerů, kdy při vysoké zátěži sítě čeká packet v bufferu nepoměrně déle než při nízkém provozu, což může způsobit chyby v synchronizaci až v řádu milisekund. [4]

Hardwarové časové značky tento problém eliminují. Značkovací jednotka síťového rozhraní je mezi MAC a fyzickou vrstvou a umožňuje tak přečíst časovou značku hned jak packet přijde na rozhraní, případně ji zapsat těsně než packet rozhraní opustí. Díky tomu je pak jediný faktor ovlivňující přesnost synchronizace zpoždění přenosové cesty.

PTP tedy funguje i v síti obsahující běžný hardware, který tento protokol nepodporuje. Přesnost synchronizace bude ale výrazně nižší, zejména bez hardwarových časových značek u OC zařízení.

1.2.5 Topologie

PTP topologie je nezávislá na fyziské topologii sítě ve které běží. Hodiny jsou uspořádány do stromové struktury, kde kořenem je vybraný Grandmaster. Každá další úroveň hierarchie je oddělena od té vyšší pomocí hraničních hodin. Listy stromu jsou jednotlivé OC Slaves. Na obrázku 1.3 je příklad tříúrovňové topologie s dvěma hraničními a jedněmi transparentními hodinami. První BC se synchrinizují podle vybraného Grandmastera, druhé BC podle prvních. Ostatní OC vždy podle předřazených BC. Transparentní hodiny (v E2E režimu) nejsou z pohledu synchronizačních zpráv vidět, proto nemají ani Master/Slave režim portů. Druhá verze protokolu zavádí alternativní topolo-



Obrázek 1.3: Příklad topologie s BC a TC.

gii, vice v sekci 1.3.3

Pro výběr Grandmastera používá PTP algoritmus nazvaný BMCA (Best Master Clock Algorithm), který automaticky vyhledá zařízení s nejkvalitnějším zdrojem času v rámci PTP domény a to nastaví jako Master. Kvalita hodin se určí podle technologie fyzických hodin zařízení (stabilita oscilátoru) a třídy. Třída se určuje podle zdroje, ze kterého zařízení bere časovou informaci, v nejvyšší třídě jsou zařízení připojené k atomovým hodinám, ve druhé nejvyšší zařízení využívající signál GPS, další třídy jsou určené vzdáleností od zařízení s jedním z těchto dvou zdrojů času. Pokud je potřeba pouze relativní synchronizace v rámci sítě, bez externího zdroje času, pak na třídě Grandmastera v podstatě nezáleží. [8]

Je možné ručně nastavit prioritu hodin pro BMCA, případně přímo některé hodiny definovat jako Preferred Grandmaster - potom nikdy nebudou v režimu Slave - nebo Slave Only - potom nikdy nebudou vybrány jako Grandmaster. To je užitečné hlavně u záložních Grandmaster hodin, které jsou v režimu Passive dokud je hlavní Grandmaster v pořádku a při výpadku je zajištěno že budou pomocí BMCA vybrány do jeho role.

1.3 Komunikace

1.3.1 Zprávy

PTP používá pro komunikaci tzv. *Zprávy (messages)*. Jsou dvě kategorie zpráv, události (event messages), u kterých se zaznamenává časová značka odeslání/přijetí, např Sync nebo Delay_Req a obecné (general messages), u kterých čas odeslání/přijetí není důležitý, např. Follow_Up a Delay_Resp. Všechny zprávy mají stejnou PTP hlavičku a dále payload podle druhu zprávy (většinou jedna nebo více časových značek), formát zpráv je více rozebrán v sekci 2.4. Druhy zpráv jsou následující:

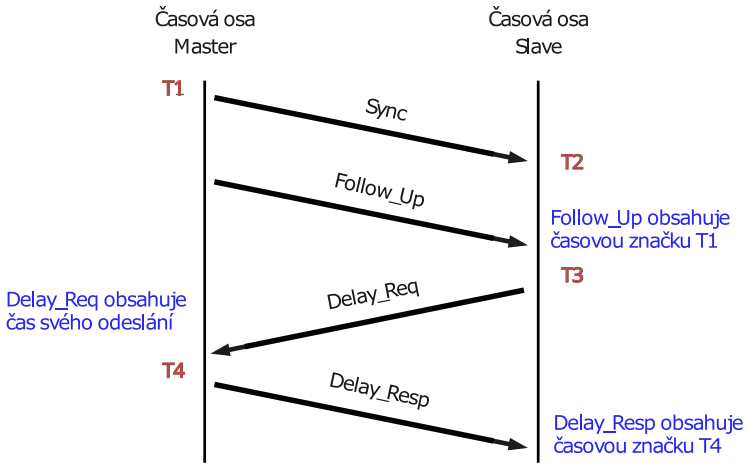
- Announce - zprávy BMC algoritmu, řeší sestavení hierarchie hodin a výběr Grandmastera
- Sync, Follow_Up - synchronizační zprávy vysílané Masterem v pravidelných intervalech
- Delay_Req, Delay_Resp - delay request/response, slouží k měření zpoždění přenosové cesty v E2E režimu
- Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up - slouží k měření zpoždění přenosové cesty v P2P režimu
- Management, Signalling - Signalizační a pomocné zprávy

1.3.2 Průběh synchronizace

O vlastní synchronizaci se starají dva mechanismy, které běží paralelně, nezávisle na sobě. První z nich představují zprávy Sync a Follow_Up. Zprávy Sync jsou pravidelně vysílány Masterem a obsahují jeho časovou značku v době odeslání zprávy. Aby tato časová značka byla dostatečně přesná, je třeba hardwarové značkování 1.2, protože je třeba zapisovat do právě odesílané zprávy. Pokud zařízení hardwarové časové značky nepodporuje, má Master možnost poslat druhou, přesnější časovou značku ve zprávě Follow_Up, která následuje za každou Sync. Tento způsob je definovaný jako TWO-STEP synchronizace a poskytuje vyšší přesnost za cenu vyšší zátěže sítě (2 zprávy místo jedné). Oproti tomu ONE-STEP synchronizace zprávy Follow_Up nepoužívá a má tak menší nároky síť i procesor Mastera. U zařízení bez HW časových značek je ale TWO-STEP způsob nutný k zajištění přijatelné přesnosti.

Druhý mechanismus je měření zpoždění přenosové cesty. Existují dva způsoby jak toto provést, implicitně se využívá End-to-End způsob definovaný v první verzi protokolu, který neklade žádné speciální požadavky na topologii sítě. Každé Slave zařízení si měří zpoždění cesty mezi sebou a Masterem. K tomuto účelu slouží zprávy Delay_Req a Delay_Resp. Zařízení pošle delay request a poznamená si časovou značku odeslání. Master, který delay request obdrží pošle delay response, do které vloží časovou značku příchodu delay requestu.

Průběh synchronizace je znázorněn na obrázku 1.4. Na začátku má Master čas TM a Slave čas $TS = TM + OFF$, kde OFF je offset vůči času Mastera. Master odešle v čase $T1$ synchronizační zprávu, kterou Slave v čase $T2$ přijme, čas $T2$ si zaznamená a čeká na Follow_Up. Po obdržení Follow_Up zná Slave časové značky $T1$ a $T2$. Podle $T1$ si posune své hodiny tak, aby $TS = TM - (Master- > Slavedelay)$ a v čase $T3$, který si opět zaznamená, pošle delay request. Master zařízení obdrží delay request, poznamená si časovou značku jeho přijetí $T4$ a tu odešle v delay response zprávě. Po přijetí delay response má slave k dispozici všechny čtyři časové značky nutné k vypočtení



Obrázek 1.4: PTP synchronizace v End-to-End režimu.

posunutí času vůči Masterovi.

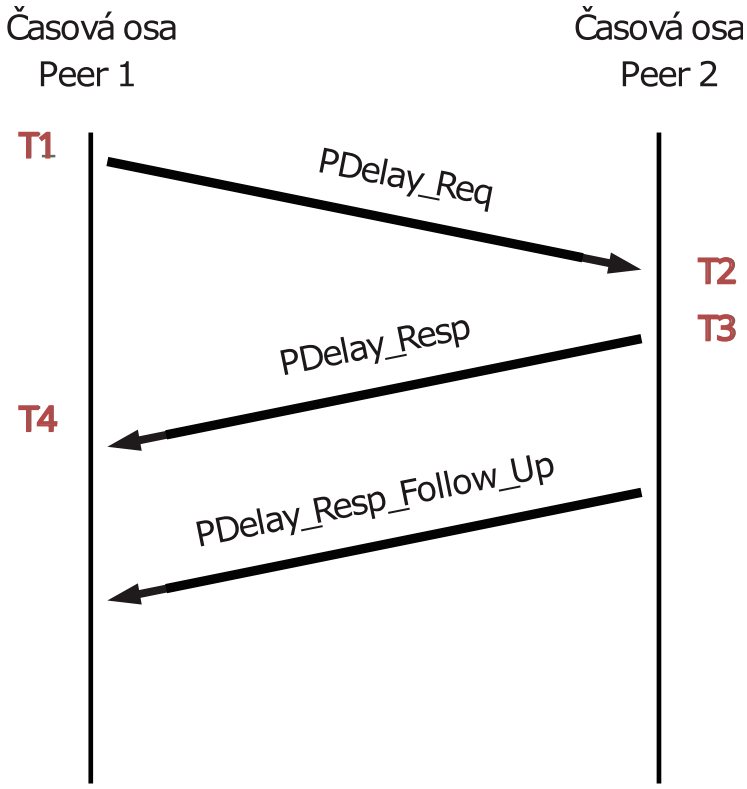
Nejprve spočte průměrnou dobu zpoždění cesty (Mean Path Delay) $MPD = ((T2 - T1) + (T4 - T3))/2$ a pomocí ní vypočte posunutí času vůči Masterovi jako $OFF = T2 - T1 - MPD$ Stále platí, že $TS = TM + OFF$, slave tedy nyní zná přesný čas:

$$TS = TM + T2 - T1 - MPD$$

1.3.3 Režimy komunikace

Základní způsob komunikace byl definován v první verzi protokolu a počítá pouze s běžnými a hraničními hodinami. [2] Každá úroveň stromové struktury PTP topologie má jedno Master zařízení, podle kterého se synchronizují všechna ostatní zařízení způsobem popsaným výše v sekci 1.3.2. Druhá verze protokolu tento způsob komunikace rozšiřuje pouze o možnost přidání transparentních hodin v E2E režimu, [6] které z pohledu synchronizačního mechanismu nejsou vidět a pouze přidávají časovou korekci do procházejících PTP zpráv, viz 1.1.

Druhá verze protokolu ale také definuje alternativní režim komunikace založený na transparentních hodinách v P2P módu. V tomto režimu je síť složena pouze z OC a P2P TC. Synchronizační zprávy se posílají stále stejně, rozdíl je v měření zpoždění přenosové cesty. Každé zařízení (včetně Mastera) si měří zpoždění přenosové cesty ke každému dalšímu pripojenému zařízení. K tomuto účelu používá podobný mechanismus jako je E2E synchronizace znázorněný na obrázku 1.5. První zařízení pošle Pdelay_Req zprávu a zaznamená si časovou značku odeslání T1. Druhé zařízení tuto zprávu přijme a zaznamená si čas



Obrázek 1.5: PTP synchronizace v Peer-to-Peer režimu.

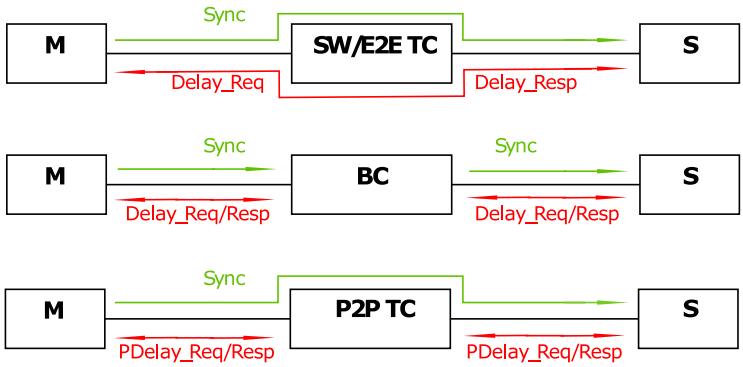
přijetí T2. Tento čas pak pošle zpět prvnímu zařízení pomocí Pdelay_Resp. Dále pošle ještě časovou značku odeslání Pdelay_Resp T3 ve zprávě Pdelay_-Resp_Follow_Up. První zařízení přijme Pdelay_Resp_Follow_Up v čase T4 a v ten okamžik už zná všechny časové značky potřebné k výpočtu zpoždění cesty mezi zařízeními.

$$MPD = ((T2 - T1) + (T4 - T3))/2$$

Takto si TC v P2P módu změří zpoždění na všech svých portech. Při průchodu synchronizační zprávy (Sync nebo Follow_Up) pak v její hlavičce upraví pole correctionField tak, že k hodnotě připočte zpoždění své cesty. Když taková zpráva dojde k Slave zařízení, už v sobě nese informaci o zpoždění cesty a Slave si tedy rovnou může spočítat časový posun vůči Master zařízení takto:

$$OFF = T2 - T1 - correctionField$$

Jako časy $T1$ a $T2$ je zde chápano odeslání Sync zprávy Master zařízením, resp. přijetí Sync zprávy Slave zařízením.



Obrázek 1.6: Funkce PTP síťových prvků v různých režimech synchronizace. M = Master, S = Slave, SW = Běžný switch, ostatní zkratky viz A

Předpokladem pro použití tohoto režimu je, že každý port každého zařízení musí být připojen maximálně k jednomu portu jiného zařízení, topologie tedy už není stromová struktura ale matice tvořená pouze běžnými hodinami a transparentními hodinami v P2P módu. Výhodou je eliminace nepřesností, které do systému vnáší použití hraničních hodin, nevýhodou je ale právě nutnost použití TC zapojených specifickým způsobem, nelze ho tedy použít (bez způsobení nepřesnoti synchronizace) pokud je v cestě běžný síťový prvek bez podpory PTP.

Rozdíly mezi chováním různých druhů hodin v různých režimech komunikace jsou vidět na obrázku 1.6.

Kapitola 2

Návrh a realizace

2.1 Vymezení problematiky

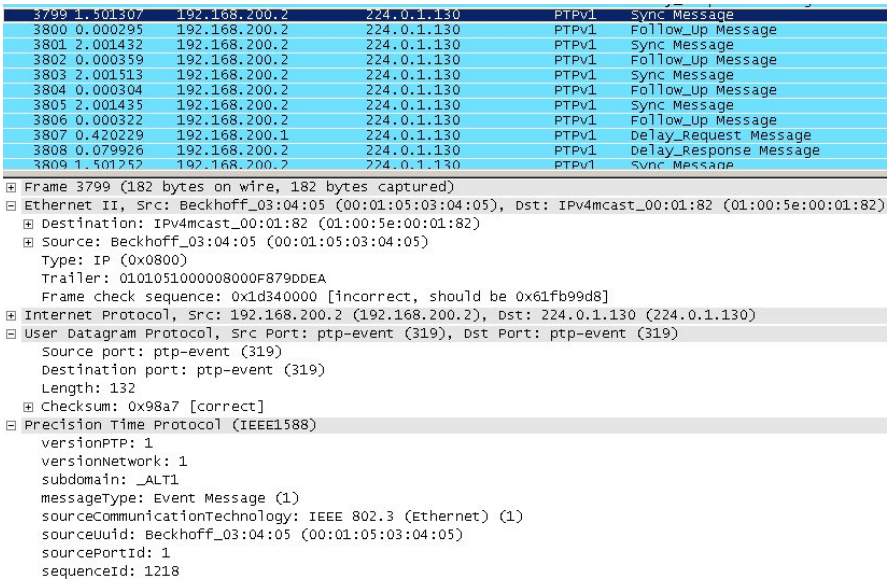
Monitor musí umět v reálném čase zobrazovat seznam PTP zařízení v síti, stav jejich hodin a parametry komunikace mezi nimi. Tyto parametry komunikace jsou například průměrné zpoždění cesty, četnost zasílání delay request a delay response zpráv a odchylka hodin jednotlivých zařízení od lokální reference.

Vzhledem k tomu, že PTP nemá vestavěné žádné diagnostické funkce, které by umožnily třetí straně získat informace o průběhu synchronizace některého ze Slave zařízení, jedinou možností je odposlouchávat komunikaci na síti, analyzovat posílané PTP zprávy a z nich dostávat potřebné informace. Z principu funkce TCP/IP sítí takové monitorování nepůjde uskutečnit v Peer-to-Peer režimu komunikace, omezím se tedy na režim End-to-End a na analýzu jednoho ze segmentů sítě (jedna úroveň PTP hierarchie), který bude vždy obsahovat jedno (nebo i více) Master zařízení a jedno nebo více Slave zařízení.

Monitor bude mít také možnost stát se přímo PTP klientem a poskytnout tak přesnější a podrobnější informace o Slave zařízení na kterém běží a také o Master zařízení, ke kterému je synchronizován. K tomuto účelu bude nejlepší použít existující implementaci PTP klienta jako démona pro linux a analyzovat jeho komunikaci.

2.2 Existující nástroje

Nástroje pro zachytávání a analýzu síťového provozu již samozřejmě existují, jdou sice použít v rámci PTP protokolu, poskytují však pouze informace o konkrétní PTP zprávě, kterou program zrovna odchytil. Požadované parametry komunikace by se daly zjistit z historie odchycených zpráv, to ale nesplňuje podmínku zobrazování informací v reálném čase.



Obrázek 2.1: Follow_Up zpráva zachycená programem Wireshark.

2.2.1 Wireshark

Dobrým příkladem programu z této kategorie je packet sniffer Wireshark, což je velmi robustní nástroj pro analýzu síťového provozu od linkové vrstvy až po aplikační. Wireshark má mnoho modulů pro analýzu aplikačních protokolů do hloubky, bohužel pro PTP žádný takový modul neexistuje. Lze tedy získat jen informace o konkrétní zprávě, jako je její zdrojová IP adresa, typ zprávy, verze PTP protokolu, sekvenční číslo a samozřejmě přenášená časová značka, viz 2.1. Tato data sama o sobě neposkytnou potřebné informace o jednotlivých zařízeních v síti, pomohou mi ale se s protokolem lépe seznámit a navrhnout mechanismus jak potřebné informace získat.

Přestože tedy Wireshark nezastane funkci požadovaného nástroje, výborně poslouží při jeho vývoji a testování.

2.3 Knihovna PCap

Knihovna PCap představuje open source API umožňující jakékoli aplikaci číst odeslané a přijaté packety přímo ze síťového adaptéru. Pokud je síťový adaptér v promiskuitním módu, lze takto zachytávat veškerou komunikaci v příslušném segmentu sítě (kolizní doméně), kde se zařízení nachází. Na této knihovně je postaveno mnoho nástrojů pro analýzu sítě včetně programu Wireshark a bude sloužit i jako základ mého PTP monitoru.

Knihovna byla vytvořena pro jazyk C a lze ji tedy i v C++ volat přímo,

bez použití wrapperu. Pro účely této práce postačí několik klíčových funkcí.

2.3.1 pcap_lookupdev()

```
char * pcap_lookupdev( char * errbuf )
```

V případě, že uživatel nezadá při startu programu síťové rozhraní, na kterém chce monitor provozovat, tato funkce najde první dostupné rozhraní. Funkce vrací jméno nalezeného rozhraní, parametrem je string buffer pro případné chyby.

2.3.2 pcap_lookupnet()

```
int pcap_lookupnet( const char * device, bpf_u_int32 * netp,
    ↪ bpf_u_int32 * maskp, char * errbuf )
```

Slouží k získání základních informací o rozhraní „device“. Do netp a maskp se uloží adresa sítě (ne zařízení) a maska podsítě.

2.3.3 pcap_open_live()

```
pcap_t * pcap_open_live( const char * device, int snaplen, int
    ↪ promisc, int to_ms, char * errbuf )
```

Tato funkce připraví síťové rozhraní „device“ k zachytávání packetů. Paramatr snaplen udává maximální délku packetů, které budou zachytávány, parametr promisc přepne síťový adaptér do promiskuitního módu a parametr to_ms udává timeout čtení packetů v milisekundách. Funkce vrací deskriptor, který využívají ostatní funkce jako odkaz na zařízení.

2.3.4 pcap_compile() a pcap_setfilter()

```
int pcap_compile( pcap_t * p, struct bpf_program * fp, const
    ↪ char * str, int optimize, bpf_u_int32 netmask )
int pcap_setfilter( pcap_t * p, struct bpf_program * fp );
```

Knihovna PCap má vestavěný mechanismus filtrování packetů. Filtrovat lze na základě mnoha kritérií, například zdrojová/cílová IP nebo MAC adresa, port, rozhraní... PTP implicitně využívá UDP porty 319 (event zprávy) a 320 (general zprávy), budu tedy chtít zachytávat pouze komunikaci na těchto portech. Při spuštění monitoru lze specifikovat jiné porty než implicitní.

Filtry se zapisují pomocí speciálního „jazyka“, každý filtr je nutné před použitím zkompilovat pomocí pcap_compile(). První argument je deskriptor zařízení získany pomocí pcap_open_live(). Druhý parametr je ukazatel na strukturu, do které bude uložen zkompilovaný filtr. Parametr str je string pro kompilaci, poslední dva parametry není třeba vysvětlovat.

Po zkompilování lze filtr aplikovat na příslušné zařízení pomocí druhé funkce pcap_setfilter(). Prvním argumentem je opět deskriptor zařízení, druhým je ukazatel na zkompilovaný filtr.

2.3.5 pcap_loop()

```
int pcap_loop( pcap_t * p, int cnt, pcap_handler callback,
    ↪ u_char * user )
```

Nakonec funkce, která provádí vlastní odchytávání packetů, dokud jich neodchytne cnt, případně není zavolána funkce pcap_breakloop(). V mém případě funkce běží stále, dokud není program ukončen. První parametr je opět deskriptor zařízení, parametr callback je ukazatel na funkci, která se zavolá pro každý zachycený packet. Poslední parametr je potom ukazatel pro uživatelská data, který se předá jako první parametr funkci callback. Tento ukazatel využívám pro předání objektu CMon, který představuje vlastní monitor, stará se o parsování zpráv a zobrazování dat.

2.4 Režim Monitor

V režimu monitor program využívá knihovnu PCap k zachytávání PTP zpráv typu Sync, Follow_Up, Delay_Req a Delay_Resp, data z těchto zpráv ukládá do vnitřní struktury a následně z nich získává potřebné informace, které zobrazuje uživateli.

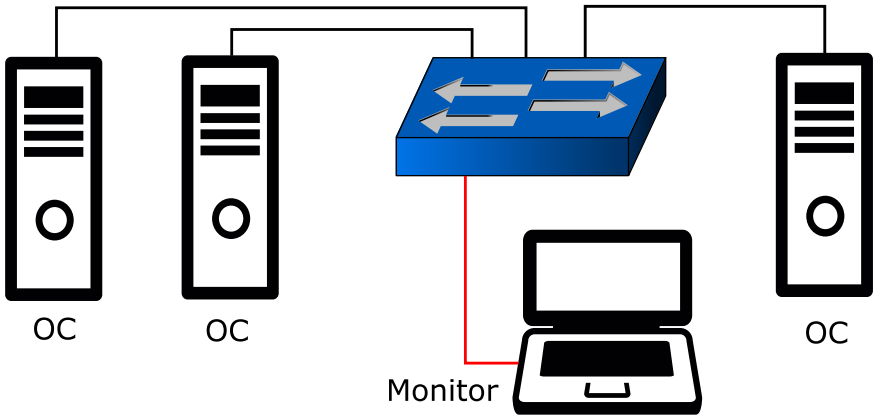
2.4.1 Možnosti

Aby mohl monitor odchytávat potřebné zprávy, musí k nim mít fyzický přístup. Pokud je síť uzpůsobena pro P2P režim, každé zařízení vidí pouze svého nejbližšího souseda, monitor spuštěný na některém z takto zapojených zařízení by viděl Sync a Follow_Up zprávy Mastera, ale zprávy typu Pdelay pouze od svých bezpostředních sousedů, neměl by tedy jak zjistit čas Slaves v síti. V P2P režimu tedy monitor nelze použít a není tedy ani třeba implementovat zachytávání zpráv typu Pdelay.

V E2E režimu je situace jiná. Monitor se spustí na počítači připojeném do sítě podle obrázku 2.2. Vyobrazený switch může z hlediska PTP představovat BC, TC nebo běžný stwitch bez podpory protokolu, ve všech případech bude funkce monitoru stejná. Na switchi je třeba nastavit zrcadlení všech portů (Port Mirroring) na port, kde je připojený zmíněný počítač, aby monitor viděl veškerý provoz, který switchem prochází.

2.4.2 Princip funkce

Takto zapojený monitor tedy kromě Sync a Follow_Up zpráv vidí i Delay_Req a Delay_Resp zprávy týkající se všech Slaves v síti. Tyto čtyři typy zpráv odchytává a dále zpracovává.



Obrázek 2.2: Zapojení do sítě v režimu monitor.

Při zachycení Sync zprávy monitor aktualizuje záznam o příslušném Master zařízení (v případě první Sync zprávy z daného zdroje vytvoří nový záznam) a zobrazí přibližný čas tohoto zařízení (zpoždění přenosové cesty je zatím neznámé). Pokud PTP používá synchronizaci ve dvou krocích, monitor před aktualizací záznamu čeká na příslušnou Follow_Up zprávu. Párování zpráv se provádí pomocí jejich sekvenčních čísel.

Při zachycení Delay_Req zprávy si monitor uloží její časovou značku a vytvoří záznam o novém Slave zařízení. Následně čeká na odpověď v podobě Delay_Resp zprávy od Mastera. Pokud tuto zachytí, spočte zvoleným způsobem (viz sekce 2.4.3) čas příslušného Slave zařízení a zobrazí ho.

Záznamy o zařízeních se nemažou, zobrazuje se vždy poslední známý čas každého zařízení, při každém výpisu se provádí korekce podle času, který uplynul od zachycení poslední Delay_Req zprávy, v ideálním stavu by se tedy zobrazené časy jednotlivých Slave zařízení měly co nejvíce blížit lokální referenci.

2.4.3 Výpočet času

Model synchronizace v E2E režimu je znázorněn na obrázku 1.4. Monitor je vždy zapojen tak, že vidí veškeré informace, které si Master a příslušní Slaves vyměňují, může tedy vypočíst čas standardním způsobem podle vztahu

$$TS = TM + T2 - T1 - MPD \text{ kde } MPD = ((T2 - T1) + (T4 - T3))/2.$$

Nastává však jeden zásadní problém, časová značka $T2$ reprezentující čas, kdy dané Slave zařízení přijalo synchronizační zprávu se nikdy nepřenáší po síti. Monitor jí tedy není schopen žádným způsobem zjistit. Je tedy třeba spočítat zpoždění přenosové cesty jinak a uživatel programu by měl zvolit nejvhodnější způsob na základě svých znalostí o konkrétní síti. Možnosti jsou:

- Použít $T3$ místo $T2$ - implicitní způsob výpočtu, zanesená chyba je rozdíl $T3 - T2$.
- Zobrazit $T3$ přímo jako čas Slave zařízení - tento způsob spoléhá na to, že Slave funguje správně.
- Vlastní Delay_Req zpráva - monitor se zeptá Mastera pomocí vlastní Delay_Req zprávy. Výsledné zpoždění ale bude odpovídat cestě mezi Masterem a monitorem, ne mezi Masterem a daným Slave. Vhodný způsob pokud je monitor připojen blízko Slave a rozdíl je malý nebo dokonce zanedbatelný.
- Známé zpoždění - uživatel přímo zadá monitoru zpoždění přenosové cesty. Hodí se pokud chceme zkoumat konkrétní Slave, o kterém tuto informaci známe, časy ostatních zařízení budou ale opět nepřesné, v závislosti na tom, jak moc se liší zpoždění jejich příslušných cest od toho zadaného.

Každý ze způsobů představuje jistý kompromis a žádný není absolutně přesný. Z tohoto důvodu obsahuje program druhý režim - Klient, který sice analyzuje jen jedno PTP slave zařízení v síti, ale bez chyb v přesnosti času.

2.4.4 Implementace

2.4.4.1 Struktura PTP zprávy

Každá PTP zpráva se skládá z hlavičky o délce 34 B a těla. V případě ptp zpráv které monitor zachytává, tedy Sync, Follow_Up, Delay_Req a Delay_Resp vždy bezprostředně za hlavičkou následuje časová značku o délce 10 B, kde prvních 6 B udává sekundy a poslední 4 B udávají nanosekundy od počátku epochy. Z pohledu synchronizačního modelu 1.4 nesou zprávy Sync a Follow_Up časovou značku T1, kde zpráva Follow_Up by měla mít její přesnější verzi. Zpráva Delay_Req nese časovou značku T3 a zpráva Delay_Resp časovou značku T4.

Struktura PTP hlavičky je vidět v tabulce 2.4.4.1. Zajímavá jsou pole typ zprávy, příznaky, časová korekce a sekvenční číslo. Důležitá je také zdrojová IP adresa packetu, který PTP zprávu obsahuje. Cílová adresa je multicast.

2.4.4.2 Zachycení zpráv

Odchytávání jednotlivých packetů zajišťuje třída *CSniffer*, která obaluje funkce knihovny PCap zmíněné výše v sekci 2.3. Při vytváření instance *CSniffer* se jejímu konstruktoru předá instance třídy *CMon*, která se stará o ukládání a zobrazování záznamů o zařízeních v síti. Dalším předávaným parametrem je jméno síťového rozhraní, na kterém se budou packety odchytávat (např. eth0). Pokud není uvedeno, vybere se první dostupné pomocí funkce *pcap_lookupdev*.

Tabulka 2.1: Struktura hlavičky PTP zprávy

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	délka	začátek
Rezervováno				Typ zprávy				1	0
Rezervováno				Verze PTP				1	1
Délka zprávy								2	2
Číslo domény								1	4
Rezervováno								1	5
Příznaky								2	6
Korekce času								8	8
Rezervováno								4	16
Identita zdrojového portu								10	20
Sekvenční Číslo								2	30
Control Field								1	32
Log Message Interval								1	33

Tabulka 2.2: Struktura Sync zprávy

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	délka	začátek
Hlavička								34	0
Časová značka T1 - čas odeslání této zprávy								10	34

Tabulka 2.3: Struktura Follow_Up zprávy

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	délka	začátek
hlavička								34	0
Přesný čas odeslání předchozí Sync zprávy								10	34

Tabulka 2.4: Struktura Delay_Req zprávy

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	délka	začátek
hlavička								34	0
Čas odeslání této zprávy ze Slave zařízení								10	34

Tabulka 2.5: Struktura Delay_Resp zprávy

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	délka	začátek
hlavička								34	0
Čas přijetí Delay_Req zprávy serverem								10	34
SourcePortIdentity z odpovídající Delay_Req zprávy								10	44

Pokud byla instance *CSniffer* úspěšně vytvořena, je možné zahájit odchytávání packetů zavoláním metody *CapturePackets*. Jejím jediným parametrem je řetězec, který definuje filtrování zachycených packetů. Řetězec je ve formátu vyžadovaném funkcí *pcap_compile*. Zachytává pouze provoz na portech protokolu PTP - UDP 319 a 320. Je možné zvolit jiné porty pomocí parametrů **-e** (pro *event messages*, defaultní port je 319) a **-g** (pro *general messages*, defaultní port je 320).

Pro každý zachycený packet se zavolá metoda *pktCallback*, která provede jeho dekapsulaci. Tato funkce při zachycení packetu dostane ukazatel na data, což je začátek ethernetového rámce. Z hlavičky rámce zkontroluje, zda odpovídá typ obsahu a vybalí z něj IP packet. U něj zkontroluje zda je kompletní, má správnou délku a jedná se o první fragment případného fragmentovaného packetu (PTP zprávy nejsou nikdy delší než MTU – 1500 B – a nebudou tedy nikdy fragmentované). Následně z packetu vybalí UDP datagram a z něj dále PTP zprávu. Z PTP hlavičky zjistí, o jaký typ zprávy se jedná a podle toho volá příslušnou metodu asociovaného *CMon* objektu.

2.4.4.3 Ukládání záznamů

O další zpracování zachycených PTP zpráv se stará třída *CMon*. Konstruktor *CMon* má dva parametry, prvním je řetězec představující název síťového rohzhranní, na kterém monitor běží. Druhý parametr určuje způsob výpočtu času jednotlivých *Slave* zařízení. Číselné hodnoty jsou přiřazeny takto:

1. Použije se časová značka v Delay_Req zprávě ($T3$) místo $T2$
2. Použije se časová značka v Delay_Req zprávě ($T3$) přímo jako aktuální čas příslušného *Slave* zařízení
3. Monitor se zeptá na zpoždění cesty pomocí vlastní Delay_Req zprávy
4. Uživatel zadal konstantní zpoždění cesty

Ukládání záznamů řeší přímo příslušné metody pro zpracování zpráv, které jsou volány z funkce *pktCallback* zmíněné výše. Každá z těchto metod má stejné dva parametry, prvním z nich je ukazatel na strukturu *my_ptp*, která představuje zachycenou PTP zprávu, druhý parametr je řetězec obsahující zdrojovou IP adresu. Každé zařízení je reprezentováno pomocí struktury *THost*, zařízení jsou ukládána pomocí kontejneru *std::map*.

Metoda *ParseSync* se volá při zachycení zprávy Sync. Pomocí knihovny *std::chrono* je vytvořena časová značka, která se považuje za čas odchycení zprávy a používá se jako offset při výpisu záznamů. Pokud zařízení s danou zdrojovou IP adresou v seznamu není, vytvoří se nová instance *THost*. Uloží se časová značka vytvořená na začátku metody a sekvenční číslo přijaté zprávy. Funkce dále zjistí, zda je použita synchronizace v jednom nebo dvou krocích

přečtením příslušného příznaku z pole *flags* PTP hlavičky a pokud je použita synchronizace v jednom kroku, uloží rovnou i časovou značku obsaženou v Sync zprávě spolu s rozdílem mezi ní a lokálním časem. Pokud je použita synchronizace ve dvou krocích, monitor čeká na příslušnou Follow_Up zprávu.

Metoda *ParseFollow_Up* je volána po zachycení Follow_Up zprávy. Metoda ověří, jestli zařízení s danou IP adresou existuje v seznamu. Pokud ne, neudělá nic. Pokud ano, zkontroluje, zda je toto zařízení vedeno jako *Master* a sekvenční číslo jeho poslední Sync zprávy odpovídá sekvenčnímu číslu právě přijaté Follow_Up zprávy. V případě že ano, zapíše k němu časovou začku obsaženou v přijaté Follow_Up zprávě spolu s rozdílem mezi ní a lokálním časem.

Metoda *ParseDelay_Req* zpracovává zprávy typu Delay_Req. Pokud byla zpráva přijata od neznámého zařízení, přidá se toto zařízení do seznamu jako *Slave*. Pokud již v seznamu existuje, pouze se inkrementuje čítač požadavků na zpoždění od tohoto zařízení. Metoda dále uloží sekvenční číslo přijaté zprávy, časovou značku, kterou zpráva obsahuje a přidá záznam do historie požadavků na zpoždění od tohoto zařízení *m_DRHistory*. Proměnná *m_est* struktury *THost* se u záznamů o *Slave* zařízeních používá jinak, místo ní je čas přijetí poslední Delay_Req zprávy uložen na konci vektoru *m_DRHistory*.

Metoda *ParseDelay_Resp* se stará o zpracování zpráv Delay_Resp a zároveň o výpočet časů jednotlivých *Slave* zařízení. Metoda nejprve přečte sekvenční číslo přijaté zprávy a zjistí, jestli byla zachycena odpovídající Delay_-Req zpráva. Pokud ne, metoda končí. Pokud ano, metoda vyhledá záznam o zařízení ke kterému zpráva patří. Následně přečte časovou značku přijaté Delay_Resp zprávy a poznamená si ji jako $T4$. Dále přečte ze záznamu o příslušném zařízení časovou značku Delay_Req zprávy s odpovídajícím sekvenčním číslem a poznamená si ji jako $T3$. Zbývá najít časovou značku $T1$. Metoda vyhledá příslušné *Master* zařízení v seznamu pomocí zdrojové IP adresy přijaté Delay_Resp zprávy. Jako $T1$ si monitor poznamená časovou značku obsaženou v poslední Sync/Follow_Up zprávě odeslané tímto zařízením. Následuje výpočet $T2$ podle zvoleného způsobu, viz 2.4.3. Vypočteý čas *Slave* zařízení se zapíše do proměnné *m_time* struktury *THost*. Spolu s ním se zapíše kdy byl tento čas naposled spočítán – položka *m_est* – a rozdíl oproti lokálnímu času – *m_clock_difference*

2.4.4.4 Zobrazení informací

K průběžnému výpisu záznamů o všech známých zařízeních používá třída *CMon* metodu Report. Každé volání této metody vyčistí okno terminálu pomocí escape příkazu
033[2J
033[1;1H, tekže se přepisují pouze časové informace. V horním pruhu je zobrazeno jméno síťového rozhraní na kterém monitor běží a aktuální lokální čas. Pod tímto pruhem je potom zobrazen seznam aktivních PTP zaříení. U

každého zařízení je vidět jeho IP adresa, druh (*Master* nebo *Slave*), čas a rozdíl času oproti lokálnímu. U *Slave* zařízení je dále zobrazen čas zachycení poslední Delay_Req zprávy, počet Delay_Req zpráv a průměrná doba mezi dvěma Delay_Req zprávami.

Metoda si po zavolání vytvoří časovou značku pomocí knihovny *std::chrono* a zjistí si aktuální čas pomocí funkce *CurrNS*. Poté vyčistí terminálové okno a vypíše vrchní pruh. Metoda dále projde všechny záznamy o zařízeních, vypíše jejich IP adresu a typ a nastaví si časový offset *dur_ns*, podle toho kolik nanosekund uplynulo od poslední zachycené Sync, případně Delay_Resp zprávy, podle typu zařízení. Veškeré časové značky jsou před výpisem převedeny do formátu $DD.MM.YYHH : MM : SS.NS$ pro lepší čitelnost. Doba uplynutá od zachycení zprávy do jejího výpisu reprezentována offsetem *dur_ns* je vždy připočtena ke zobrazované časové hodnotě, takže není třeba se jí při čtení hodnot zabývat.

2.5 Režim Klient

V režimu Klient program využívá open source implementaci Ordinary Clock pro linux – PTPd (Precision Time Prtotocol Daemon). Tohoto démona lze nakonfigurovat tak, aby zaznamenával statistiky a jiné užitečné informace do stavového souboru (Status File), který je potom zpracován programem. Vzhledem k tomu, že klient má přímo k dispozici všechny čtyři časové značky potřebné pro E2E synchronizaci času, budou zobrazované hodnoty přesnější než v režimu Monitor.

2.5.1 PTPd

2.5.1.1 Popis

Při hledání vhodné softwarové implementace PTP pro linux jsem narazil na dva projekty, linuxptp a PTPd. Oba projekty jsou aktivně vyvýjeny a splňují všechny funkční požadavky pro tuto aplikaci. PTPd má ale několik významných výhod. První z nich je zmíněný stavový soubor, který se dynamicky aktualizuje za běhu démona, takže je ideální pro účel monitorování protokolu bez nutnosti zasahovat do kódu démona. Druhou výhodou je, že PTPd má připravené instalační balíčky pro mnoho linuxových distribucí, instalace je tedy velice pohodlná a není nutné kvůli ní rekompilovat součásti systému. Z těchto důvodů jsem si pro Klient režim mé aplikace vybral právě PTPd.

Nevýhodou naopak je, že starší verze PTPd mohou mít trochu jinou sktrukturu konfiguračního souboru a jiné významy některých přepínačů při volání z příkazové řádky. Klíčové záznamy ve stavovém souboru jsou ale ve všech verzích, které jsem měl možnost vyzkoušet identické.

V rámci programu se o Klient režim stará třída *CClient*. Její kontruktor má jako první parametr cestu ke stavovému souboru PTPd a jako druhý parametr název síťového rozhraní, na kterém klient poběží.

2.5.1.2 Konfigurace PTPd

Instalace PTPd probíhá standardně přes balíčkovací systém příslušné linuxové distribuce. Ve své síti používám distribuci Debian a Linux Mint, který z něj vychází, nainstaluji tedy PTPd pomocí *apt-get install ptpd*. Po instalaci lze démon spustit jako služba na pozadí příkazem *service ptpd start*, nebo jako běžný proces (*ptpd*).

Program se sám stará o spouštění PTPd s potřebnými parametry, je ale třeba mít konfigurační soubor **ptpdslave.conf** v adresáři, ze kterého je program spouštěn. Tento konfigurační soubor zajišťuje správné chování PTPd pro potřeby programu, jako je například logování do stavového souboru. Konfigurační soubor není třeba nijak upravovat, i přesto zde zmíním důležité direktivy:

- ptpengine:interface - síťové rozhraní, na kterém klient poslouchá. Direktivu upraví program při spouštění démona.
- ptpengine:preset - nastaveno na *slaveonly*, šablona pro ordinary clock v režimu slave.
- ptpengine:transport - nastaveno na *ipv4*, PTP přímo po Ethernetu není programem podporováno.
- ptpengine:ip_mode - nastaveno na *multicast*, program pracuje pouze v milticast. Funkce v unicast režimu by měla být možná, nebyla ale ověřena.
- ptpengine:delay_mechanism - nastaveno na *E2E*, program nepodporuje P2P režim měření zpoždění.
- global:log_status - nastaveno na *Y*, zapne či vypne používání stavového souboru.
- global:status_file - cesta ke stavovému souboru, implicitně */var/run/ptpd2.status*.

Kompletní struktura konfiguračního souboru je popsána v manuálových stránkách PTPd a samotný soubor je dobře okomentován.

2.5.1.3 Stavový Soubor

Jak bylo zmíněno výše, PTPd si během své činnosti o sobě ukládá nejrůznější informace do dynamicky aktualizovaného stavového souboru (Status File). Tento soubor se aktualizuje v intervalu nastaveném v konfiguračním souboru (viz výše). Pro potřeby aplikace PTPMon jsou zajímavé následující parametry:

- Best Master IP - IP adresa aktuálního nejlepšího Mastera. Pokud žádný neexistuje, záznam v souboru není.
- Host Info - informace o procesu, obsahuje PID, pomocí kterého program PTPd ukončuje.
- Sync Mode - druh synchronizace (v jednom nebo dvou krocích)
- Offset from Master - offset (rozdíl časů) oproti Masterovi
- Mean Path Delay - průměrné zpoždění přenosové cesty
- Sync Received - počet přijatých Sync zpráv
- Follow_Up Received - počet přijatých Follow_Up zpráv
- Delay_Req Sent - počet odeslaných Delay_Req zpráv
- Delay_Resp Received - počet přijatých Delay_Resp zpráv

Stavový soubor je zpracován metodou *ParseStatusFile*. Tato metoda představuje jednoduchý parser, který čte soubor řádek po řádku a podle příslušných klíčových slov ukládá potřebné údaje pro pozdější vypsání.

2.5.2 Zobrazení aktuálního stavu

O výpis získaných údajů se stará metoda *Report*, která funguje podobně, jako stejnojmenná metoda třídy *CMon*. Na začátku si vyčistí terminálové okno a zobrazí vrchní pruh se jménem síťového rozhraní a aktuálním časem získaným pomocí knihovny *std::chrono*. Pokud není k dispozici žádné Master zařízení, je zobrazena zpráva „Searching for a master...“ Pokud klient Master zařízení našel, zobrazí jeho IP adresu a čas a dále údaje vyjmenované v sekci 2.5.1.3 v uvedeném pořadí.

2.6 Uživatelské rozhraní

Program pracuje v prostředí příkazového řádku s textovými výstupy. Zobrazený výstup se ale dynamicky aktualizuje, podobně jako například u Unixových nástrojů *prstat* nebo *top*.

Program se spouští běžným způsobem z aktuálního adresáře příkazem *./ptpmon* a vyžaduje jeden povinný parametr, kterým je název síťového rozhraní na němž poběží. Dále je třeba zvolit režim operace přepínači **-m** pro režim Monitor a **-c** pro režim Klient. K dispozici je i několik dalších přepínačů:

- **-t** - zvolí metodu výpočtu času Slave zařízení, viz sekce 2.4.4.3, pouze v režimu Monitor.

- **-d** - nastaví pevné zpoždění cesty, má smysl pouze pro metodu 3 výpočtu času Slave zařízení v režimu Monitor.
- **-o** - program použije již běžící PTPd proces místo startování vlastního. Povinným parametrem je cesta k alternativnímu stavový souboru, který bude použit jako zdroj informací. Přepínač má smysl pouze v režimu Klient.
- **-e** - pomocí něj lze nastavit vlastní port pro zprávy událostí (Event messages), pouze v režimu Monitor.
- **-g** - pomocí něj lze nastavit vlastní port pro běžné zprávy (General messages), pouze v režimu Monitor.

Po spuštění program zobrazí několik zpráv o své aktuální činnosti, a pokud je úspěšně spuštěn, zobrazí se hlavní okno. Pokud nastane při inicializaci chyba, program skončí výjimkou a zobrazí příslušnou chybovou hlášku. Hlaví okno má vždy u vrchního okraje pruh, kde je zobrazeno rozhranní, na kterém program běží a aktuální čas z lokálního zdroje, získaný pomocí knihovny *std::chrono*. Pod tímto pruhem je zpráva o vybraném režimu funkce a dále už údaje o monitorované síti. UI programu lze vidět v akci na screenshotech v sekci 3.

Kapitola 3

Testování

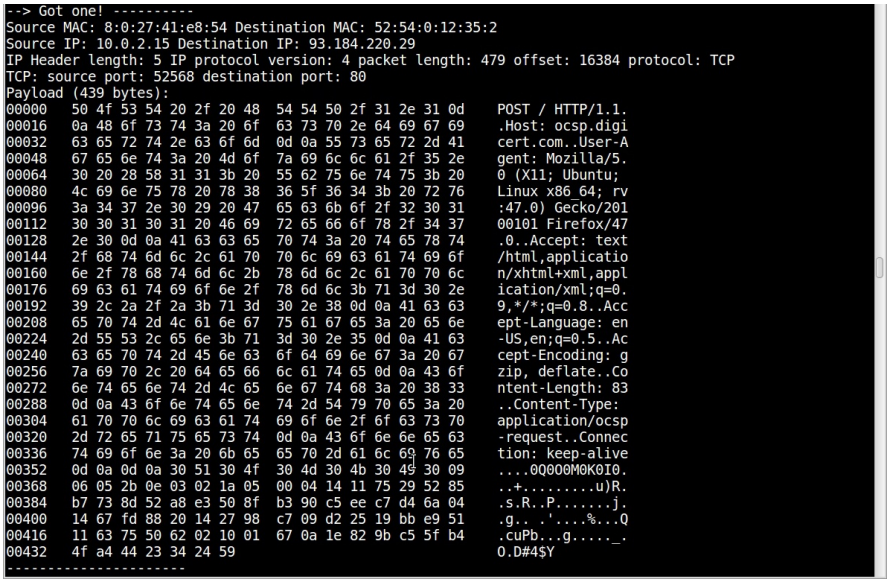
V této kapitole je aplikace prakticky předvedena. Nejprve jsem ověřil základní funkčnost zachytávání packetů a zkontroloval chování programu v různých chybových stavech. Dále jsem pomocí aplikace zkoumal rozdíly v chování PTP protokolu ve třech různých síťových prostředích. První z nich byla domácí bezdrátová Wi-Fi síť, druhé metalická Ethernetová síť a třetí rovněž Ethernetová síť, ale s použitím síťových prvků s podporou PTP.

3.1 Ověření funkčnosti

Zachytávání packetů jsem testoval spuštěním programu se zapnutými debug výpisy, kdy se každý zachycený packet vypsal na standardní výstup. Souběžně s aplikací byl spuštěn program Wireshark, aby byla možnost zachycené packety porovnávat. Počet packetů zachycených aplikací se shodoval s počtem packetů zachyceným programem Wireshark, obsah packetů rovněž odpovídal. Na obrázku 3.1 je vidět zachycený HTTP požadavek.

Aplikace byla dále otestována v různých nestandartních situacích, s nesprávnými vstupy, chybnými přepínači, nedostatečnými právy ke spuštění, zápisu na disk a přístupu k síťové kartě. Zkoušel jsem i různé druhy chybných packetů a náhlé odpojení od sítě. Ve všech případech aplikace korektně skončila výjimkou. Jediný problém byl se spouštěním aplikace v režimu Klient, kde je potřeba nastavit rozhranní používané PTPd démonem pomocí konfiguračního souboru. Pokud aplikace spouští vlastní PTPd proces, příslušnou direktivu konfiguračního souboru si edituje podle parametru zadaného z příkazové řádky. Pokud ale uživatel zvolí použití externího PTPd procesu, je třeba správné nastavení v konfiguračním souboru zkontrolovat.

Ověření funkčnosti tříd CMon a CClient bylo provedeno na virtuální síti s použitím programu Oracle VirtualBox. Byly spuštěny dva virtuální stroje, oba s operačním systémem Linux Debian a démonem PTPd. Na prvním stroji byl PTPd nastaven jako Master-only, na druhém byla jako Slave použita aplikace v režimu Klient. Zároveň v rámci druhého procesu běžela aplikace v režimu



Obrázek 3.1: HTTP požadavek zachycený třídou CSNiffer.

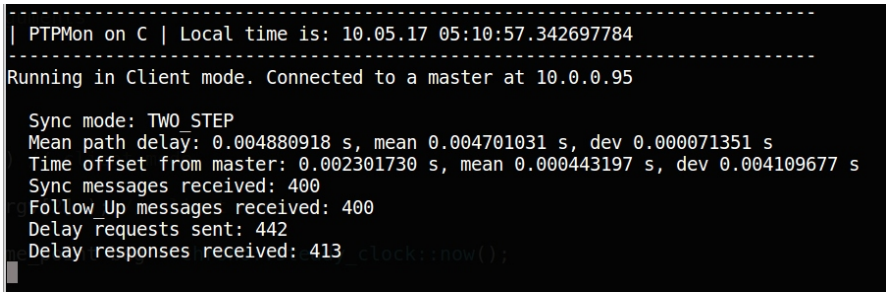
Monitor. Všechny části aplikace fungovaly správně, pouze mi chyběla možnost, jak dohledat změny v posunutí (offsetu) časů monitorovaných zařízení oproti lokálnímu času. Aplikaci jsem tedy rozšířil o jednoduché logování, kdy se aktuální offset vždy před změnou zapíše do souboru. Díky tomu, že celá síť byla emulována v operační paměti, byly parametry PTP komunikace velmi dobré, screenshoty s konkrétními hodnotami nejsou zobrazeny, jelikož byly téměř identické s hodnotami naměřenými v Ethernetové síti, viz **??**

3.2 Měření v různých sítích

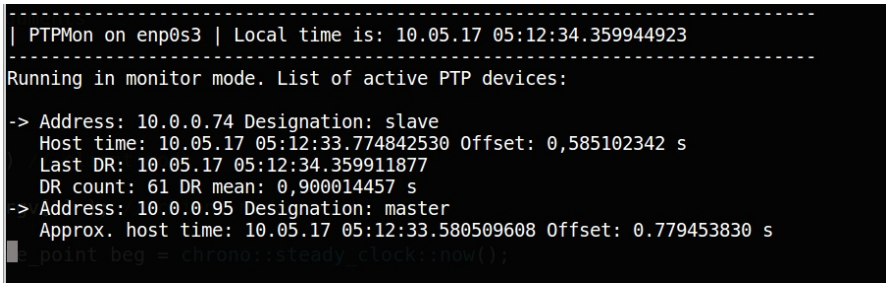
Po kontrole funkčnosti samotné aplikace již bylo možné použít ji k účelu, pro který vznikla - monitorování PTP provozu v síti. Záměrem bylo zjistit, jaký vliv bude mít technologie fyzické vrstvy a zátěž sítě na synchronizaci času.

3.2.1 Domácí Wi-Fi síť

První měření proběhlo v rámci mé domácí sítě. Konfigurace PTP byla velmi podobná virtuální síti v předchozí sekci, funkci Grandmaster clock zastal PTPd démon běžící na notebooku s operačním systémem Linux Mint. Na druhém notebooku se stejnými parametry byly spuštěny dvě instance aplikace, jedna v Klient režimu, druhá v režimu Monitor. Všechna zařízení byla připojena k SOHO Wi-Fi routeru Zyxel vmg1312-b30b pomocí bezdrátové technologie IEEE 801.11n. Síť byla úmyslně nadprůměrně zatížena, kromě zmí-



Obrázek 3.2: Aplikace v režimu Klient, bezdrátová domácí síť

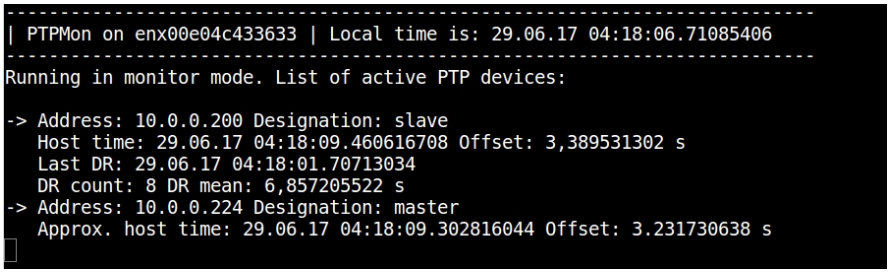


Obrázek 3.3: Aplikace v režimu Monitor, bezdrátová domácí síť

něných dvou notebooků byl k síti připojen počítač se spuštěnou online hrou, hlasovým komunikačním programem TeamSpeak a video streamem směrem do internetu, byl spuštěn stream HD videa z NAS serveru do televize a zahájeno kopírování několika velkých souborů pomocí protokolu SMB/CIFS. Na druhou stranu je tato síť provozována v málo zarušeném prostředí v rodinném domku.

Výpočet zpoždění cesty byl nastaven na implicitní možnost, kdy se místo časové značky T2 používá značka T3. Výsledky jsou vidět na obrázcích 3.2 a 3.3. Na prvním obrázku běží program v režimu Klient a zobrazené hodnoty se pohybují v očekávaných mezích. Podle zkoumaného stavového souboru PTPd je po přijetí 400 Sync zpráv průměrný offset od Master hodin kolem 400μs, ale hodně kolísá, což je vidět na posledním spočteném offsetu 2.3ms a zobrazené směrodatné odchylce.

Na druhém obrázku je vidět stejná situace, tentokrát ale aplikace běží v režimu Monitor. Z hodnot lze vyčíst, že sledovaný Slave posílá Delay_-Req zprávy přibližně jednou za sekundu. Zobrazený offset přes 500 ms byl způsoben špatným nastavením lokálního času. Po opravě nastavení tak, aby byl referenční čas shodný s časem Mastera však zobrazované hodnoty stále výrazně kolísaly, offset se pohyboval v rozmezí jedné až desítek milisekund.



Obrázek 3.4: Měření v Ethernetové síti

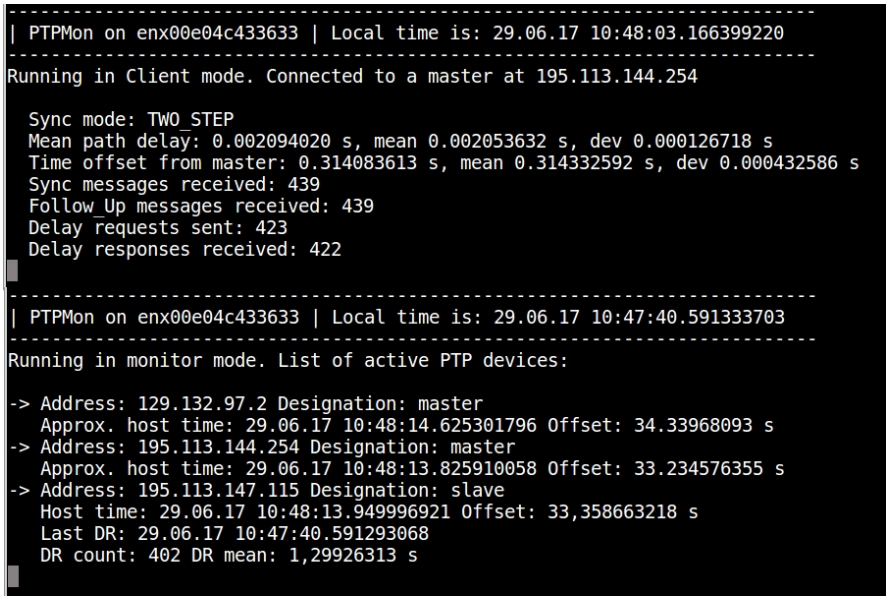
3.2.2 Ethernetová síť

Druhý test proběhl v metalické síti využívající 100BaseTX Ethernet. Opět byl jako Grandmaster použit PTPd démon běžící na notebooku s OS Linux Mint a monitor běžel na notebooku se stejnou konfigurací. Jako Slave byl tentokrát použit jednodeskový počítač RaspberryPi s OS Linux Raspbian, což je klon Debianu určený pro architekturu ARM, na které je RaspberryPi postaven. Na tomto počítači byl také spuštěn démon PTPd ve Slave-only režimu. Protože jsem v době testování neměl k dispozici switch s možností zrcadlení portů, byl místo něj použit Ethernetový hub 3Com OfficeConnect 16. Síť byla zatížena pouze testovaným PTP protokolem, očekával jsem tedy mnohem lepší výsledky než v předchozím testu.

Screenshot z tohoto měření je na obrázku 3.4. Oproti prvnímu testu hodnoty o poznání méně kolísaly, velký offset je opět způsoben tím, že lokální čas monitoru není nijak synchronizován s časem Grandmastera. Změny v offsetu byly ale v řádu maximálně desítek μs. Četnost odesílání Delay_Req zpráv je také o poznání nižší, skoro sedminová oproti prvnímu testu. Synchronizace v této síti je tedy podle očekávání přesnější a stabilnější.

3.2.3 Ethernetová síť s hardware podporou PTP

Poslední měření bylo provedeno ve spolupráci s vedoucím práce RNDr. Ing. Vladimírem Smotlachou, Ph.D. v laboratoři Cesnet v Praze. K dispozici mi byl průmyslový switch Hirschmann MS20 s hardware podporou ptp (viz 1.1) a dedikované Grandmaster hodiny Meinberg. Jako Slave a zároveň monitor sloužil již několikrát zmíněný notebook s OS Linux Mint. Slave funkcionalitu zajišťovala aplikace v režimu Klient. Grandmaster hodiny byly se switchem propojeny pomocí Gigabit Ethernetu a switch s monitorem potom pomocí 100BaseTX Ethernetu. Díky hardwarové podpoře by měly být naměřené hodnoty nejlepší ze všech tří testovacích prostředí, zatížení sítě bylo rovněž minimální. Také bylo vidět, jak se aplikace chová v síti s více Master zařízeními. Screenshoty z tohoto testu jsou na obrázku 3.5.



Obrázek 3.5: Měření v Ethernetové síti s HW podporou PTP

Navzdory očekávání se zjištěné hodnoty téměř nelišily od hodnot naměřených v Ethernetové síti s běžnými síťovými prvky. Offset byl opět velice stabilní a četnost Delay_Req zpráv dokonce vyšší než u druhé testované sítě. Tento fakt je dán jednak tím, že klient, stejně jako monitor byl spuštěn na notebooku se síťovou kartou bez podpory hardwarových časových značek (viz 1.2) a hlavně tím, že monitor pracuje s PTP zprávami na čtvrté vrstvě OSI modelu, zprávy jsou tedy enkapsulované ve třech úrovních, PTP zpráva -> UDP datagram -> IP packet -> Ethernetový rámec, což zvyšuje režii nutnou na rozbalení zachycených packetů, která je řádově vyšší než zpoždění způsobené čekáním packetu v paměti switche.

Závěr

V průběhu své bakalářské práce jsem se důkladně seznámil s protokolem PTP a úskalími přesné synchronizace času v počítačové síti. Na základě zjištěných skutečností jsem se pokusil najít co nejlepší řešení problému, kterým bylo vytvořit aplikaci schopnou v reálném čase sledovat provoz v protokolu PTP. Otestoval jsem několik existujících nástrojů pro síťový dohled postavených na knihovně PCap, žádný z nich ale nevyhovoval specifickým potřebám této práce. Řešením tedy bylo vytvořit si vlastní aplikaci.

Výsledkem je program PTPmon. Program je napsán v jazyce C++ a určen pro systém Linux. PTPmon má dva režimy, Monitor a Klient, v režimu monitor využívá knihovnu PCap k zachytávání packetů ze sítě. Tyto pakety filtruje a analyzuje a získané informace v reálném čase zobrazuje v terminálovém okně. Při vývoji jsem narazil na problém s výpočtem času jednotlivých klientů, který se mi podařilo vyřešit pouze kompromisem, uživatel programu má na výběr ze čtyř způsobů výpočtu a musí se sám rozhodnout, který je pro testovanou síť nejvhodnější. Režim Klient využívá existující softwarovou implementaci PTP pro linux - démona PTPd. Tento démon si statistiky o svém provozu zaznamenává do stavového souboru. PTPmon v režimu Klient tento soubor čte a vybírá z něj informace, které opět zobrazuje dynamicky v okně terminálu. Zobrazené informace jsou přesnější než v režimu Monitor, zkoumán je však pouze jeden PTP klient. Aplikace PTPmon byla použita pro ověření funkce PTP ve třech různýc síťových prostředích a svůj účel splnila. Svou bakalářskou práci tedy považuji za úspěšnou.

Téma PTP a synchronizace času vůbec se ukázalo mnohem rozsáhlejším, než jsem zpočátku čekal, ale také velice zajímavým. Určitě zbývá prostor pro další vylepšování, vhodným rozšířením by mohlo být například zachytávání PTP zpráv na druhé vrstvě OSI, tedy podpora enkapsulace přímo do Ethernetových rámců, aplikace v současném stavu podporuje pouze zprávy posílané přes UDP protokol. Také by bylo zajímavé rozšířit aplikaci o možnost synchronizace lokálního času, který aplikace považuje za přesnou referenci, s PTP Grandmasterem, pro přesnější zobrazení posunutí času jednotlivých zařízení

oproti referenci, užitečné by také mohlo být extenzivnější logování.

Literatura

[1] Synchronizace v distribuovaných řídicích systémech: Precision Time Protocol (PTP) podle IEEE 1588. *Automa, časopis pro automatizační techniku' [online]*, 2010, [cit. 2017-06-30]. Dostupné z: `http://automa.cz/cz/casopis-clanky/synchronizace-v-distribuovanych-ridicich-systemech-precision-time-protocol-ptp-podle-ieee-1588-2010_02_40557_5798/`

[2] IEEE Standards Association: *1588-2002 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems [online]*. [cit. 2017-06-30]. Dostupné z: `https://standards.ieee.org/findstds/standard/1588-2002.html`

[3] Eidson, J. C.: *Measurement, Control, and Communication Using IEEE 1588*. Springer London, první vydání, ISBN 978-1-84628-250-8.

[4] Weibel, P. H.: IEEE 1588 Standard for a Precision Clock Synchronization Protocol. [online], 2012.

[5] Zurawski, R.: *Industrial Communication Technology Handbook*. Druhé vydání, ISBN 9781138071810.

[6] IEEE Standards Association: *1588-2008 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems [online]*. [cit. 2017-06-30]. Dostupné z: `https://standards.ieee.org/findstds/standard/1588-2008.html`

[7] Breuer, J.: *Synchronizace času v distribuovaných heterogenních měřicích ařídicích systémech*. Dizertační práce, České Vysoké učení Technické v Praze, Fakulta Elektrotechnická, 2016.

[8] Implementing IEEE 1588v2 for use in the mobile backhaul. *Calnex Solutions technical brief [online]*, 2014, [cit. 2017-06-30]. Dostupné

z: http://www.chronos.co.uk/files/pdfs/cal/TechnicalBrief-IEEE1588v2PTP.pdf

Příloha **A**

Seznam použitých zkratek

API Application programming interface - Knihovna funkcí, tříd a/nebo datových struktur umožňující programátorovi jednoduše využívat prostředky jiných programů nebo např. prostředky operačního systému

ARM Architektura procesorů určených především pro vestavné systémy

BC Boundary Clock - Druh PTP hodin

BMC/BMCA Best Master Clock algorithm - Algoritmus pro automatický výběr nejlepších hodin do role *Grandmastera*

E2E End to End - Způsob měření zpoždění přenosové cesty v rámci PTP

GPS Global Positioning System - Systém satelitů určený primárně pro geolokaci. GPS signál lze také využít jako zdroj přesného času

HD High Definition - Označení pro video ve vysokém rozlišení

HTTP Hyper-Text Transfer Protocol - Aplikační (sedmá vrstva) protokol určený pro přenos dokumentů ve formátu HTML

HW Hardware - Fyzické součásti počítače nebo sítě

IEEE Institute of Electical and Electronics Engineers - mezinárodní nezisková profesní organizace zajišťující mimo jiné standardizaci mnoha technologií. Jednou z nich je i IEEE 1588

IP Internet Protocol - Síťový protokol rodiny TCP/IP, pracuje na třetí vrstvě OSI, vyskytuje se ve verzi 4 a 6

IPv4 Internet protocol verze 4 - Nejpoužívanější síťový protokol třetí vrstvy OSI, používá 32-bitovou adresu pro jednotlivá zařízení v síti

IPv6 Internet protocil verze 6 - Rozšířená a vylepšená verze IPv4, používá 128-bitovou adresu.

LAN Local Area Network - Síť malého rozsahu, pokrývající většinou oblast jedné budovy nebo skupiny budov

MAC Media Access Control address - Adresa linkové (druhé) vrstvy, unikátní 48-bitový identifikátor síťového zařízení

MPD/TMPD Mean Path Delay - Průměrné zpoždění přenosové cesty

MTU Maximum Transmission Unit - Maximální velikost datové jednotky (např. IP packetu), kterou je daný komunikační protokol schopný přenést v celku (v rámci jedné transakce)

NAS Network-Attached Storage - Úložiště dat dostupné ze sítě

OC Ordinary Clock - Druh PTP hodin

OS Operační systém - Základní programové vybavení počítače, stará se o přidělování systémových zdrojů dalším programům

OSI Open Systems Interconnection - Teoretický model rozdělující komunikaci v počítačových sítích do sedmi vrstev, každá vrstva má svůj účel a může využívat nižší vrstvy pro své potřeby

P2P Peer to Peer - Způsob měření zpoždění přenosové cesty v rámci PTP

PC Personal Computer - Obecné označení počítače

PID Process ID - Unikátní identifikátor procesu (programu zavedeného v paměti)

PTP Precision time protocol

PTPd Precision time protocol daemon - Název open-source softwarové implementace PTP ordinary clock

SMB/CIFS Server Message Block/Common Internet File System - Aplikační (sedmá vrstva) protokol sloužící pro sdílení souborů a zařízení v síti, nativní pro operační systémy Microsoft Windows

SOHO Small Office/Home Office - Označení pro síťový hardware určený pro domácnosti a malé kanceláře, u těchto zařízení je většinou kladen důraz hlavně na nízkou cenu

STL Standard Template Library - Knihovna algoritmů a datových struktur jazyka C++

TC Transparent Clock - Druh PTP hodin

UDP User Datagram Protocol - Síťový protokol transportní (čtvrté) vrstvy

UI User Interface - Uživatelské rozhranní

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `bin` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `doc` ..... dokumentace zdrojových kódů
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF