Önálló laboratórium beszámoló

Távközlési és Médiainformatikai Tanszék

készítette: **Schweitzer András Attila**
schweitzeraa16@gmail.com
neptun-kód: **TLEIB5**
ágazat: **Intelligens hálózatok**
konzulens: **Németh Felicián**
nemethf@tmit.bme.hu
konzulens: **Lévai Tamás**
levait@tmit.bme.hu

Téma címe: Többutas adatátvitel Media over Quic rendszerben (Media over Multipath QUIC)

Feladat:
A Media over QUIC (MoQ) egy új, még fejlesztés alatt álló protokollcsalád, amelyet az IETF szabványosít. Egyik megvalósítása a LibQuicR, amely a feladat alapjául szolgál. A feladat maga pedig nem más, mint a LibQuicR könyvtár módosítása oly módon, hogy az képes legyen kihasználni a multipath (többutas) adatátviteli képességeket, amelyeket a PicoQUIC nevű, QUIC protokollt megvalósító könyvtár biztosít. A munka során során a LibQuicR transzport rétegét úgy kell átalakítani, hogy az egy kapcsolaton belül több hálózati útvonal létesítésére és használatára legyen alkalmas a PicoQUIC multipath implementációján keresztül.

A módosításokat követően a rendszer működését egy demonstrációval igazoljuk: egy virtuális tesztkörnyezetben szimulált, több hálózati interfésszel rendelkező kliensek segítségével történik az adatátvitel, amely során aktívan használjuk a multipath képességeket. A demonstráció során egy olyan forgatókönyvet vizsgálunk, ahol az egyik aktív útvonalat szándékosan megszakítjuk. A cél annak bemutatása, hogy a LibQuicR – az új multipath támogatással – beavatkozás nélkül képes fenntartani az adatfolyam továbbítását, minimális késleltetéssel és megszakítással.

Tanév: 2024/25. tanév, II. félév

1. A laboratóriumi munka környezetének ismertetése, a munka előzményei és kiindulási állapota

1.1. Bevezető

Napjainkban az internetes adatforgalom túlnyomó részét különféle médiatartalmak – elsősorban videók és élő közvetítések – teszik ki. Egyes kutatások szerint szerint az internetes forgalom 65%-a ilyen típusú tartalom lehet [1]. Ezek hatékony és megbízható továbbítása azonban komoly technológiai kihívást jelent, különösen akkor, ha a felhasználói élményt is figyelembe vesszük: az alacsony késleltetés, a megszakításmentes lejátszás és a biztonságos adatátvitel alapvető elvárások.

E kihívásokra jött létre válaszként a QUIC protokoll, amely UDP-alapú, alacsony késleltetésű, titkosított és megbízható adatátvitelt tesz lehetővé. Erre épülve fejlődik jelenleg a Media over QUIC (MoQ) szabvány, amely kifejezetten médiatartalmak hatékony továbbítását célozza a QUIC képességeit kihasználva.

Ezzel párhuzamosan a felhasználói eszközök – különösen a mobiltelefonok és laptopok – egyre gyakrabban rendelkeznek több hálózati interfésszel (például Wi-Fi és 5G). Mégis gyakran tapasztalható kapcsolatmegszakadás, hosszú töltési idők vagy éppenséggel elérhetetlenné váló szolgáltatások, ha az egyik kapcsolat megszakad vagy instabillá válik. A többutas adatátvitel lehetőséget kínál ezen problémák áthidalására azáltal, hogy párhuzamosan több hálózati útvonalat használ az adatok továbbítására, növelve ezzel a robusztusságot és az elérhető sávszélességet.

A jelen munka célja ezen két technológia – a médiatartalmakra optimalizált QUIC-alapú továbbítás és a multipath kommunikáció – egyesítése. A cél egy olyan rendszer bemutatása, amely képes több útvonalat kihasználva, élő adatfolyamot hatékonyan és megbízhatóan továbbítani. Ez nemcsak a hálózati erőforrások jobb kihasználását segíti elő, hanem hozzájárul a végfelhasználói élmény javításához is.

1.2. Elméleti összefoglaló

1.2.1. A QUIC protokoll

Az internetes multimédiaátvitel technológiai fejlődése során egyre nagyobb igény mutatkozik olyan protokollokra, amelyek képesek rugalmasan és hatékonyan kezelni a valós idejű adatfolyamokat, még változékony és megbízhatatlan hálózati körülmények között is. [2] A QUIC protokoll – mint UDP-alapú, titkosított és kapcsolatorientált transzportprotokoll – alapjaiban újraértelmezi a hálózati kommunikáció lehetőségeit, különösen olyan kiterjesztésekkel, mint a *multipath* adatátvitel [3]. A multipath képesség lehetővé teszi több párhuzamos hálózati útvonal (**path**) egyidejű vagy váltott használatát egyetlen logikai kapcsolat keretén belül, ami különösen hasznos mobil eszközök, redundáns hálózatok vagy edge hálózati környezetek esetén.

A QUIC protokoll és a HTTP/3 működésének vizsgálatára fejlesztették ki hozzá a qlog [9] formátumot, amelynek célja, hogy strukturált és gépileg olvasható formátumban rögzítse a QUIC kapcsolatok eseményeit és metrikáit. Ez tulajodnképpen egy JSON-t generál a kapcsolat bármely végpontjában ami tartalmaz minden olyan adatot amely a kapcsolat során generálódik a QUIC rétegben, beleértve a csomagok küldését és fogadását, a kapcsolat állapotát, a késleltetéseket és a hibákat is.

1.2.2. QUIC multipath alapjai

A multipath koncepció technikai alapja, hogy a QUIC kapcsolat során nemcsak egy, hanem több **path** hozható létre, amelyeket a transzport réteg párhuzamosan vagy dinamikusan képes használni. Egy *path* definíció szerint egy adott forrás-cél IP-cím és port kombináció, azaz különböző fizikai vagy logikai hálózati kapcsolatok reprezentálása. A projektben használt PicoQUIC könyvtár egy QUIC megvalósítás, amelynek fejlődő multipath támogatása lehetővé teszi, hogy egy kapcsolat során több ilyen path aktív legyen, és az alkalmazás (quic-et megvalósító rétegében egy belső algoritmus) meghatározza, hogyan ossza meg az adatforgalmat ezek között [4].

PicoQUIC esetében a multipath kezelés alapvetően decentralizált és eseményvezérelt. A kapcsolat felépítése során egy adott interfészen elindított kapcsolat kiegészíthető további **PATH CHALLENGE / RESPONSE** (útvonal próba / útvonal válasz) üzeneteken keresztül feltérképezett útvonalakkal. Ha egy új path válik elérhetővé (például egy új IP-cím vagy interfész aktiválódik), a rendszer felismeri azt, és lehetőséget

biztosít az adatküldés ezen az útvonalon történő elindítására. A path-ek állapotát folyamatosan figyeli a protokoll (RTT, veszteség, állapotváltozás), így lehetővé válik az útvonalak közötti dinamikus váltás vagy forgalommegosztás a kapcsolatok megszakítása nélkül. Ez különösen fontos valós-idejű alkalmazásokban, mivel lehetővé teszi a megszakítás nélküli adattovábbitást, még hálózati hiba vagy mobilitás esetén is.

1.2.3. A Media over QUIC (MoQ) protokoll

A Media over QUIC (MoQ) egy új, még fejlesztés alatt álló protokollcsalád [5], amely a QUIC nyújtotta lehetőségekre építve biztosít alacsony késleltetésű, valós idejű médiatovábbítást. A MoQ rendszerében a szerepkörök három alapvető típus köré szerveződnek: a **Publisher** (szolgáltató) az, aki a médiatartalmat (például videó vagy hangfolyam) létrehozza és továbbításra bocsátja, a **Subscriber** (fogyasztó) a végponti fogyasztó, aki ezt a tartalmat fogadja és feldolgozza/lejátsza, míg a **Relay** (továbbító) köztes szereplőként funkcionál, amely a számára elérhető tartalmat hirdeti és továbbítja más résztvevők felé, jellemzően a késleltetés, terheléselosztás és elérhetőség optimalizálása érdekében. Ezek a szereplők struktúrált MoQ adatfolyamokon keresztül kommunikálnak egymással, amelyek sávokból (*track*) azon belül pedig objektumokból(*object*) épülnek fel, lehetővé téve a médiatartalom rugalmas azonosítását és replikációját.

A kidolgozás alatt álló szabvány szerint egy MoQ-alapú rendszerben a relay képes egyszerre több publisher és subscriber felé is kapcsolatot fenntartani, és akár multicast-szerű módon továbbítani a médiatartalmat. A hálózati topológia e szerepkörök között rengetegféle lehet, mivel a relayek közötti kapcsolat is támogatva van ami lehetővé teszi a tartalom replikációját és elosztását a különböző relayek között. Ezáltal a gazdag struktúrabeli opciók lehetőséget biztosítanak a tartalom dinamikus és optimalizált terjesztésére, azonban egyben igényli azt is, hogy a transzport réteg rugalmasan tudjon alkalmazkodni a változó hálózati viszonyokhoz – például egy útvonal meghibásodásához, vagy új alternatív útvonal megjelenéséhez.

1.2.4. Multipath célja a MoQ-ban

A multipath működés bevezetése ebbe az architektúrába jelentős előnyt nyújthat, különösen olyan eszközök esetén, amelyek egyszerre több hálózati interfésszel rendelkeznek (például Wi-Fi és mobil adatkapcsolat szimultán használata). A relay-ek és végpontok (subscriber, publisher) több interfésszel rendelkező környezetben történő működése során a multipath támogatás nemcsak a redundanciát növeli, hanem lehetőséget biztosít egyfajta **hálózati adaptivitásra is**, amely révén például a relay automatikusan kiválaszthatja a legjobb elérhető útvonalat egy adott irányba. Ez az architektúra ideális alapot teremt olyan rendszerek számára, ahol fontos a magas rendelkezésre állás, az alacsony késleltetés és az automatikus hibatűrés.

A MoQ protokoll jelenlegi állapota még fejlesztés alatt áll, és jelenleg egyik implementáció sem támogatja a multipath képességeket.

Bár más MoQ megvalósítások is léteznek, azért a LibQuicR könyvtárra esett a választás, mert a QUIC protokollt a PicoQUIC könyvtár segítségével implementálja, amely jelenleg is folyamatosan bővül, és multpath támogatása is követi a legújabb szabványosítási irányokat.

1.3. A munka állapota, készültségi foka a félév elején

A munka félév kezdeti állapota röviden összefoglalható azzal, hogy saját tapasztalatom nem volt sem a QUIC protokollal, sem a LibQuicR kódjával kapcsolatban, valamint a téma is új, tehát nem volt mire építeni.

2. Az elvégzett munka és az eredmények ismertetése

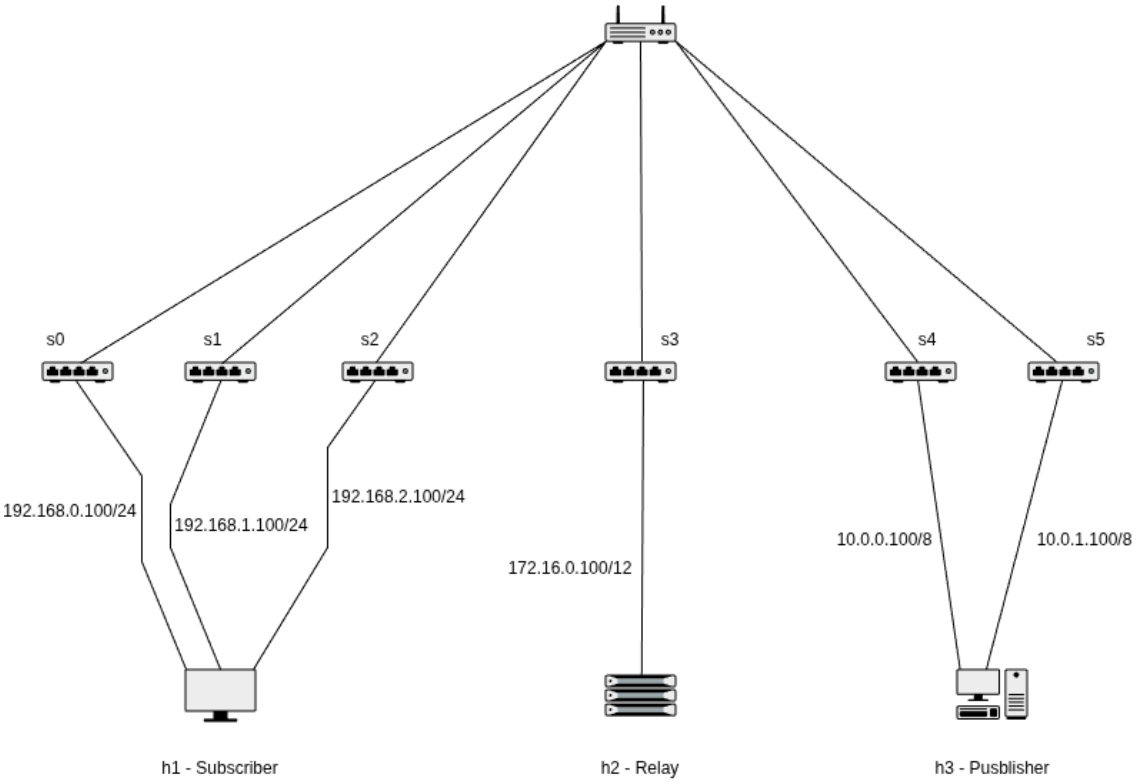
2.1. A fejlesztés és elemzés lépései egy multipath demonstrációhoz

2.1.1. Mininet szimulációs környezet

Első sorban szükséges volt egy hálózat virtualizálására alkalmas környezet, amely lehetővé teszi a különböző hálózati topológiák és viszonyok szimulálását, hiszen csak egy kontrollált környezetben lehet érdemben tesztelni a multipath működést. E célból a Mininet nevű eszközre esett a választás, mivel a használatához szükséges tudás elsajátítása nem igényelt sok időt.

A Mininet segítségével könnyen létrehozhatók virtuális gépek (továbbiakban hostok vagy h1/h2/h..), amelyek a futtató gépen létrehozott hálózati névterek, amelyeken virtuális hálózati interfészeket és azokhoz hálózati konfigurációt rendelhetünk. Ezekben a névterekben lehet futtatni a tesztelni és megfigyelni kívánt programokat. A Mininet emellett lehetővé teszi a hálózati topológia szimuláció közbeni megváltoztatását is, így kapcsolatok megszűnését is könnyen szimulálhatjuk vele. [6]

Az alábbi topológiát hoztam létre a Mininet segítségével, amellyel a multipath működését teszteltem (1. ábra):



1. ábra. A Mininetben létrehozott topológia [7]

Ahogy említettem a h1, h2, h3 jelölik a hostokat, s0-s5 jelölik a switch-eket, és r0 jelöli a routert. h1 három interfésszel rendelkezik, egyenként csatlakozik a s0, s1 és s2 switch-ekhez, azok pedig a r0 routerhez. Hasonlóan h2 is, csak egy interfésszel, a h3 pedig kettővel csatlakozik.

A hostok és a router között azért van szükség switch-ekre, mert a kapcsolatok szimuláció közbeni megszakítása problematikus, ha a megváltoztatott kapcsolat közvetlenül a host-okhoz csatlakozik. Így az alábbi topológián egy switch és a router közötti kapcsolat megszakításával könnyen lehet szimulálni a kapcsolat megszakadását.

Címzés szempontjából a hostok eltérő alhálózatokban vannak, (a h1-nek még emellett mindhárom interfésze eltérő alhálózatban van, hogy az alapértelmezett átjáró eltérő lehessen, ezzel pontosabban szimulálva a valós környezetet),

A munkát és a tesztelést felgyorsítva mindezt egy Python szkript segítségével automatizáltam, amely a mininet API-ját használja a virtuális gépek és kapcsolatok létrehozására, konfigurálására, kezelésére, beleértve a hostokon futtatott demonstrációs programokat és a Wireshark csomagelkapást is. Tehát egy script futtatásával pillanatok alatt megfigyelhető a kívánt program működése.

2.1.2. Megismerkedás a PicoQUIC könyvtárral

A féléves munka kezdetén első célom az volt, hogy alaposan megismerjem a PicoQUIC könyvtárat, mivel ez képezte a későbbi fejlesztések technikai alapját. A megismerés első lépése a **picoquicdemo** példaprogram tanulmányozása és futtatása volt, ami segített megérteni a könyvtár felépítését, működését, de különösen a multpath kezelés módjait.

Maga a pcioquicdemo felépítése elég egyszerű, olyan értelemben hogy ez egy darab C fájl, amely a PicoQUIC függvényeit használja, és a szerver, valamint a kliensek funkcióit is ellátja egyaránt.

Sajnos pont emiatt az átláthatóságot nem feltétlen egyszerűsíti meg, de összességében a munka során kiderült, hogy két dolog szükséges a multipath működéshez:

- a kezdeti multipath negotiation (egyeztetés) – ami arra szolgál, hogy a két fél közötti első kapcsolat felépítésekor megegyezzenek az összes többi transzport paraméter mellett abban is, hogy mind a két fél támogatja-e a többutas kapcsolatot
- illetve az "extra" útvonalak tényleges kiépítése

Ezek körül az utóbbit megvalósító kódrészletek könnyen fellelhetőek, mivel ezen a téren a program sok fajta opciót biztosít, amelyek közül a legegyszerűbb eshetőség az alapszintű multipath, amikor több interfész használatával több útvonalat alakítunk ki, amelyeket a PicoQUIC könyvtár automatikusan kezel.

Jelenlegi munka szempontjából ez az egyszerű multipath a leginkább releváns, mivel amíg az nem működik megfelelően, addig a komplexebb multipath forgatókönyvek nem is igazán tesztelhetők/használhatók. Szerencsére az ezt ellátó kódrészlet a picoquicdemo-ban jól elkülönül, így jó alapot is képzett a LibQuicR könyvtárban való implementálásához.

2.1.3. Működés elemzésének módjai

Munkám során elég korán előjött a kérdés, hogy hogyan tudom a PicoQUIC működését elemezni, mivel a tényleges működést nehéz folyamatában nyomon követni, hasznos lenne PicoQUIC kapcsolatot menedzselő adatfolyamot is látni.

A PicoQUIC rengeteg funkciót és metrikát biztosít a működés elemzésére, első sorban a qlog fájlok generálása lenne a leginkább logikus lépés, és a protokoll alapszintű működésének megértése szempontjából valóban hasznos a qlog, de nem minden esetben elegendő, mivel a QUIC protokoll sokféle eseményt és metrikát generál, amelyek közül sok nem feltétlenül releváns a multipath működés szempontjából és maga a multipath is megnehezíti mivel az egyetlen qlog vizualizáló amit találtam az a qvis [10] [1] volt, ami sajnos nem tudja könnyen és átláthatóan kezelni a multipath eseményeket, így ezzel nem tudtam megfelelően elemezni a kapcsolatokat.

Ezt a nehézséget kiváltandó a Wireshark program választása sokkal hasznosabbnak bizonyult, mivel a QUIC protokollt is támogatja, illetve tanulmányaim során is találkoztam már vele. Azonban ez olyan kihívást állít a qlog-gal szemben, hogy míg a qlog fájlokat a program maga generálja, tehát a titkosítást ki tudja kerülni, addig a Wireshark magukat a hálózati interfészen megjelenő csomagokat rögzíti, így a QUIC csomagok titkosítása megnehezítette a folyamatot, főleg a multipath miatt, mivel a másodlagos útvonalon küldött és fogadott csomagokat a Wireshark nem tudja hozzákötni automatikusan az első útvonal interfészén létrejött kapcsolat titkosított csomagjaihoz.

A PicoQUIC beépítetten támogatja a SSL keylog fájlok generálását a "SSLKEYLOGFILE" környezeti változó beállításával, ami segítségével fel lehet oldani a quic által használt tls titkosítást, így a Wireshark képes dekódolni a csomagokat, és megjeleníteni azok tartalmát (legalábbis a kapcsolat kezeléséért felelős quic réteg szempontjából).

Ez a megoldás lehetővé tette számomra, hogy a Wireshark segítségével generált pcapng fájlokat átkonvertáljam dekódolt formátumra az alábbi paranccsal:

[1] qvis: webes alkalmazás, amely a qlog fájlok vizualizálására szolgál, és sajnos a multipath használata könnyen eltorzítja a két oldal kapcsolatának a szinkronizációját.

```
editcap --inject-secrets tls,<keylog_file> <input_pcap_file>
  <output_pcap_file>
```

Ez a parancs a Wireshark által generált TLS titkosítást használó QUIC csomagokat tartalmazó pcapng fájlokat dekódolja, a keylog fájl segítségével, így azok bármilyen kontextusban, a keylog fájl nélkül megtekinthetőek és elemezhetőek egy Wireshark programban.

2.1.4. LibQuicR felépítése és átalakítása

A LibQuicR könyvtár természetesen egy nagyobb volumenű kihívás mint a picoquicdemo program megértése, mivel amíg az utóbbi egy darab C fájl, ami direkt használja a PicoQUIC könyvtárat, addig a LibQuicR egy sokkal komplexebb struktúrával rendelkezik, amelyben a PicoQUIC könyvtár sokkal kevésébb közvetlenül van jelen. Ennek ellenére pont azért, mert egy jelentősen nagyobb struktúráról van szó, szükségszerűen sokkal jobban elkülönülnek a különböző funkciók, így a PicoQUIC API-ját használó kódrészletek is.

Tulajdonképpen 3 funkciót kellett implementálni a LibQuicR könyvtárban, amelyek a multipath működéshez kapcsolódnak:

1. CLI argumentumok kiegészítése multipath specifikus lehetőségekkel
2. a multipath negotiation (egyeztetés) – ami arra szolgál, hogy a két fél közötti első kapcsolat felépítésekor megegyezzenek az összes többi transzport paraméter mellett abban is, hogy mind a két fél támogatja-e a többutas kapcsolatot
3. a másodlagos útvonalak tényleges kiépítése

Nyilvánvalóan ebből a második és harmadik megegyezik a korábban PicoQUIC-nél említettekkel, mivel ezek elengedhetetlenek a multipath működéshez, de ahhoz, hogy ezt érdemben lehessen használni is, szükséges volt a CLI argumentumok kiegészítése is. Ezt a picoquicdemo-ban használatos argumentumokhoz hasonlóan valósítottam meg, mivel ha már PicoQUIC-re alapozik a LibQuicR, és maga a fejlesztésem pedig annak a multipath demonstrálására szolgáló demó programjára alapozik, akkor a CLI argumentumoknak hasonlósága logikus lépés.

Relay esetén ez a CLI argumentumok kiegészítése csak egy extra argumentumot jelentett:

```
-m, --multipath
```

Ez egy szimpla kapcsoló (nem fogad extra paramétert), amely oly módon engedélyezi a multipath támogatást, hogy amikor létrejön a relay objektum QUIC traszportért felelős része, akkor azt úgy inicializálja, hogy egy bejövő kapcsolat esetén a transzport paraméterek egyeztetésénél a saját paraméterei között szerepeljen a multipath támogatás is.

Végponti kapcsolat esetén ez két kiegészítést jelentett:

- `-m, --multipath`
- `-a "...", --alt_ifaces "..."`

Itt természetesen a multipath kapcsoló ugyanúgy működik mint a relay esetén, de emellett szükséges volt egy új argumentum is, amely lehetővé teszi a felhasználó számára, hogy megadja a másodlagos interfészeket, amelyeken a multipath harmadik szükséges elemét, a másodlagos útvonalakat kiépíti. Ehhez szüksége van paraméterekre, amelyek a másodlagos interfészeket tartalmazzák, itt is a picoquicdemo-val azonos formátumot használtam, tehát a következő formátumot várja:

```
-a "<ip-cím>/<interfész száma>,<ip-cím>/<interfész száma>,..."}
```

Például a h1 host esetén ez az alábbi módon néz ki: "192.168.1.100/3,192.168.2.100/4". Látható, hogy az alapértelmezett ip címre nincs szüksége, mert azon már elkezdődött a kapcsolat, és csak azokat az interfészeket kell megadni, amelyeken a másodlagos útvonalakat kiépíti.

A megfelelő ip-cím/interfész párokat az

```
ip addr show
```

paranccsal lehet lekérdezni linuxon, amely megjeleníti a host interfészeinek ip-címét és számát.

Az eltérés a relay és a végponti kapcsolat között abban rejlik, hogy míg a relay fogadja a bejövő kapcsolatokat, addig a végponti kapcsolat esetén önállóan, a kapcsolat létrejötte után küldd PATH CHALLENGE üzeneteket a másodlagos interfészeken, hogy azok is elérhetővé váljanak a kapcsolat számára (amennyiben a multipath negotiation sikeresen megtörtént). Relay esetén több interfész használatát nem implementálja jelenleg a munkám.

A multipath negotiation megvalósulásához ki kellett egészíteni a LibQuicR-ben használt adatstruktúrákat, hogy eljussanak a multipath és az interfész információk arra a szintre, ahol a PicoQUIC könyvtár API-ját használja a program, valamint a PicoQUIC transzport objektum konstruktorát kellett kiegészíteni, és a transzport paraméterek inicializálásakor a mulitpath paraméter mellett az

```
initial_max_path_id
```

változót kellett beállítani, hogy kapcsolat létrehozása után tudjon kezelni több útvonalat is.

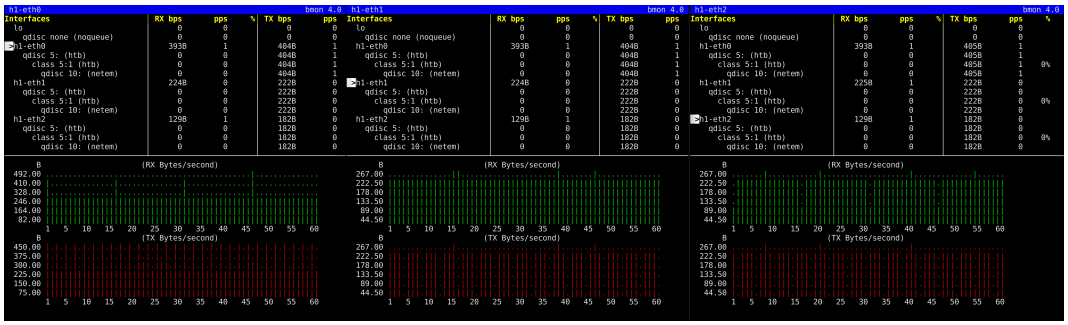
A többi útvonal kiépítése gyakorlatilag megegyező módon működik, mint a picoquicdemo esetén. A PicoQUIC

```
packet_loop_v3()
```

függvénye minden egyes csomag feldolgozása után meghív egy callback függvényt, amely a LibQuicR könyvtárban van implementálva, és ebben történik meg – ha minden körülmény adott, tehát kliensről van szó, multipath engedélyezett, és addig a pontig még nem volt új útvonal próbálva – hogy a másodlagos interfészeken PATH CHALLENGE üzeneteket küldjön. Ebben a callback függvényben történik egy segédfüggvény segítségével a másodlagos interfészeket tartalmazó sztring feldolgozása is, amely logikailag ugyancsak azonos módon működik mint a picoquicdemo esetén.

2.1.5. Elvégzett munka eredményei

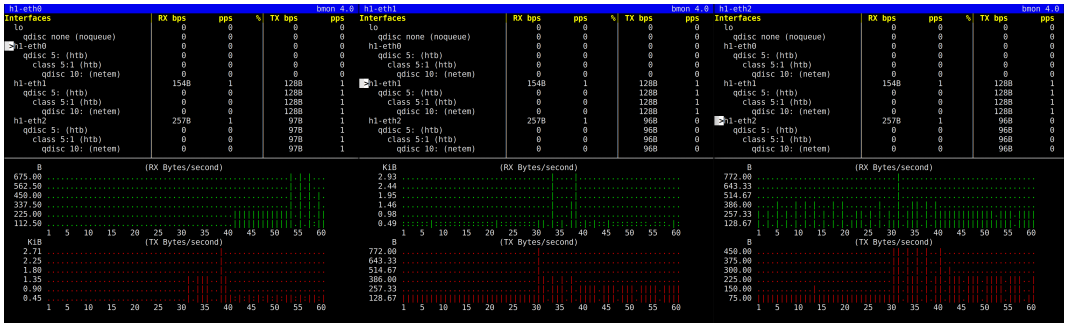
Módosításaim eredményeként a bemutatott környezetben működik a multipath kapcsolat, amely a másodlagos interfészeken keresztül adatokat küldeni és fogadni, továbbá valamely útvonal megszakadása esetén minimális késleltetéssel képes a másodlagos interfészen folytani az adatátvitelt. Maga az adatátvitel időbélyegek másdopercenkénti küldését jelenti, ami egy beépített módja a LibQuicR demó programjának.



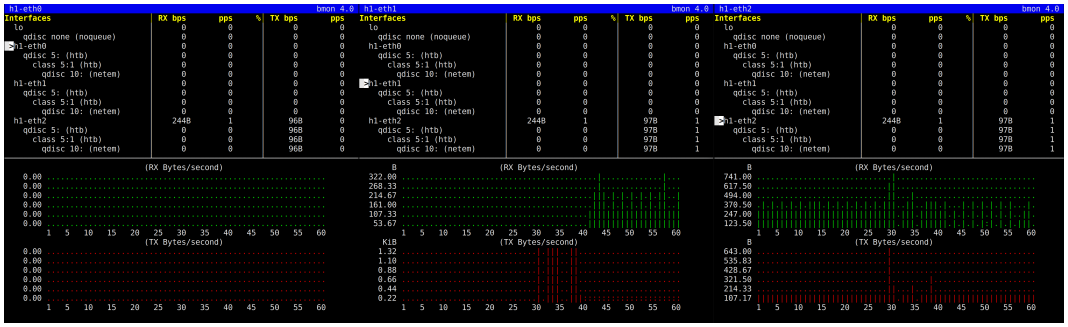
2. ábra. Interfész aktivitás 3 aktív útvonallal

2. ábra képén látható, hogy 3 aktív interfészen keresztül van folyamatos adatátvitel, majd a 3. ábrán már látható, hogy a nullás indexű interfészen nem folyik adat, mivel itt az ahhoz kapcsolódó switch és a router közötti kapcsolatot megszakítottam. Ezért a másodlagos interfészeken keresztül folytatódik az adatátvitel. Ezek nem csak azt bizonyítják, hogy a multipath implementáció kezeli a kapcsolatszakadást, de azt is, hogy akár 3 interfészt is tud hasznosítani.

Majd pedig a 4. ábra képén látható, hogy az egyes indexű interfészen is leállt az adatátvitel (ugyancsak kapcsolatszakadás miatt), de a kettes indexű interfész még mindig képes folytani a kapcsolatot.



3. ábra. Interfész aktivitás 2 aktív útvonallal



4. ábra. Interfész aktivitás 1 aktív útvonallal

Fontos megjegyezni, hogy a relay minimális kódváltoztatással képes minden alábbi szituációban működni:

- subscriber és publisher is 1-1 intefészen kapcsolódik
- csak subscriber vagy csak publisher kapcsolódik egynél több interfészen
- mind a subscirber és a publisher is több interfészen kapcsolódik

Illetve érdekes azt is megjegyezni, hogy a korábban említett

```
initial_max_path_id
```

változó a kódban 2-re van állítva, ha multipath környezetben indul el a program, de ez nem korlátozza le kapcsolatonként két interfész használatára sem a relayt, sem pedig a subscribert vagy a publishert, mint látható a 2. ábrán, ahol 3 interfész van aktívan használva.

2.2. Összefoglalás

2.2.1. Féléves feladat összefoglalása

A félév során a Quic feletti médiaátvitelt megvalósító LibQuicR könyvtárban implementáltam egy kezdetleges multipath funckionalitást, amely lehetővé teszi a relay és a végponti kapcsolatok számára, hogy több interfészen keresztül kommunikáljanak egymással, valamint ezt demonstráltam egy Mininet alapú virtuális környezetben.

2.2.2. Feladat érdekessége

Ez az első ilyen implementáció MoQ architektúrában, és bár kezdetleges multipath funckionalitást szolgáltat, de látható belőle, hogy rengeteg lehetséges irány van a továbbfejlesztésére, amelyek mindegyikéhez jó alapot képez a félév során elvégzett munka. Lehetséges folytatásai a projektnek:

- relay oldalon a másodlagos interfészek használata

- picquicdemo programban megtalálható egyéb multipath lehetőségek implementálása
  - kapcsolati azonosítók megújítása
  - NAT rebinding [2] (NAT kapcsolat újratársítás) lehetősége
  - azonos címről, de másik porttal új útvonal létrehozása
  - kapcsolat teljes migrációja a másodlagos interfészek egyikére
- relay és relay közötti kapcsolatban multipath használata
- olyan demó program készítése amely valós médiaátvitelt végez multipath használatával (videó streamelés)

2.2.3. A munka során használt eszközök

A Wireshark és a Mininet programok sok segítséget nyújtottak a munka során, mind a protokollok megismerésében és megértésében, mind pedig a munka és azon belül is a tesztelés automatizálásában, de ezeken felül haladva a világgal, próbáltam nagy nyelvi modelleket is hasznosítani a munka során, mint például a ChatGPT, vagy a GitHub Copilot, amelyek sok esetben segítettek a kód megértésében, de ugyanakkor sok esetben megmutatták, hogy mik a határaik, mivel ezek az eszközök se tudják teljes egészében átlátni a projekt struktúráját, emiatt gyakran csak tág keretek között tudtak tippeket adni, de konkrét irányokat nem tudtak mutatni. Így természetesen a legtöbbször manuálisan kellett felfedeznem a kapcsolódási pontokat a LibQuicR és a picoQUIC között, aminek eredményeként sokkal jobb rálátást kaptam a két könyvtár közötti kapcsolatra.

2.2.4. Legfontosabb eredményei a munkának

Két lényeges eredménye van a munkának:

- A LibQuicR könyvtárban a multipath működésének kezdetleges implementálása
- SSL keylog fájl generálását beépítettem a LibQuicR könytár általam módosított verziójába

Ezeket a félév végén egy GitHub pull request keretében szeretném megosztani a LibQuicR könyvtár fejlesztőivel, hogy a könyvtár további fejlesztése során egyrészt létrejöjjön egy multipath fejlesztési irány is a projektben, másrészt hogy a picoQUIC könyvtárral való együttműködést is megkönnyítse a LibQuicR könyvtár fejlesztői számára.

2.2.5. Egyéb következtetések amelyekre a munka során jutottam

A Wireshark használata során érdekes kérdés merült fel a multipath PicoQUIC implementálásával kapcsolatban, mégpedig, hogy a kapcsolat útvonalait nem feltétlen kezeli az IETF Multipath draftja szerint, mivel adott útvonal kapcsán útvonal szakadás esetén nem láttunk PATH ABANDON üzenetet, ami arra szolgálna, hogy egy már nem működő útvonalat inaktívvá nyilvánítson mind a két fél számára. Azonban olyan érdekes jelenség is előfordult, hogy A és B útvonal esetében A útvonal megszakítása után B útvonalon ment tovább az adatátvitel, de ha A útvonal újra aktívvá vált, akkor nem használta újra azt, és ezután B útvonal megszakítása után sem váltott vissza A útvonalra. Viszont mi ezt még inkább érdekessé teszi, hogy B útvonalat újra aktiválva az adatátvitel folytatódott. Tehát a PicoQUIC multipath implementációja nem feltétlenül kezel minden esetet a draft szerint, de ez nem feltétlenül probléma jelenleg, mivel még ez is fejlesztés alatt áll, illetve ez is kijelöl egy lehetséges irányt a projekt folytatására.

[2] A NAT rebinding (vagy NAT kapcsolat újtársítás) egy olyan jelenség, amikor a hálózati címfordító (NAT) eszköz új külső IP-cím–port párost rendel a klienshez, például újracsatlakozás után

3. Irodalom, és csatlakozó dokumentumok jegyzéke

3.1. A tanulmányozott irodalom jegyzéke

[1] Douglas Karr, *Live Streaming Trends and Statistics (2024)*, https://martech.zone/live-streaming-trends-statistics/

[2] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, IETF, May 2021.
https://www.rfc-editor.org/rfc/rfc9000

[3] QUIC Working Group, *Multipath Extension for QUIC, draft-ietf-quic-multipath-14, 2025. apr. 24.*, https://datatracker.ietf.org/doc/draft-ietf-quic-multipath/14/

[4] Christian Huitema / Private Octopus Inc., *PicoQUIC GitHub repository*, https://github.com/private-octopus/picoquic

[5] IETF, *Media over QUIC Transport, 2025. apr. 28.*, https://datatracker.ietf.org/doc/draft-ietf-moq-transport/11/

[6] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX)*, pages 19–24, Monterey, California, 2010. Association for Computing Machinery. https://doi.org/10.1145/1868447.1868466.

[7] Draw.io segítségével készült ábra, https://app.diagrams.net/

[8] Quicr, *LibQuicR GitHub repository*, https://github.com/Quicr/libquicr

[9] Robin Marx. qlog: A standardized schema for logging and visualizing QUIC events.
https://github.com/quiclog/qlog

[10] Robin Marx. qvis: A web-based tool for visualizing qlog files.
https://qvis.quictools.info

3.2. A csatlakozó dokumentumok jegyzéke

Saját multipath implementációm GitHub repository-ja: https://github.com/Schweitzee/libquicr/tree/multipath

SSL keylog fájl generálásához szükséges módosítások: https://github.com/Schweitzee/libquicr/tree/ssl-keylog