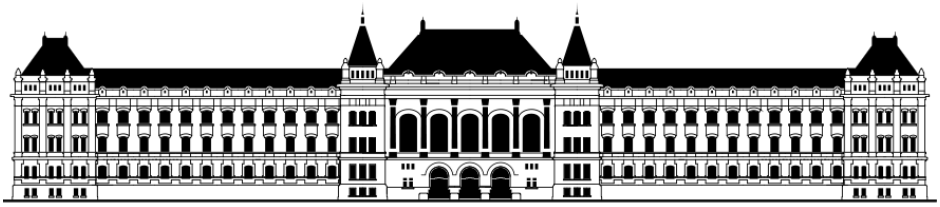


Budapesti Műszaki és Gazdaságtudományi Egyetem

Távközlési és Médiainformatikai Tanszék

Hálózati kódolás alapú médiaszórás elosztott hálózatokban

Szerzők:

Soós Ádám, Weimann András, Zalatnay Zsolt

Konzulensek:

Dr. Simon Csaba, Dr. Maliosz Markosz

Budapest, 2013.

Tartalom

1 Bevezetés

Napjainkban egyre több elosztott, heterogén és bizonytalan hálózati környezetben robusztus kommunikációt megkövetelő hálózat jelenik meg. Jellemzően egy adott területen szétszórt szenzorok és automatikus eszközökből kialakított hálózat kapcsolatai változó sávszélességűek és megbízhatatlanok.

A változatos hálózati interfésszel rendelkező okostelefonok elterjedése lehetővé tette a nagyméretű heterogén hálózatok megjelenését. A mozgásban levő fogyasztók által használt interaktív multimédia gazdag alkalmazások jelentős többletforgalmat generál, amely új tartalom elosztó mechanizmusok használatát teszi szükségessé. A hálózati kódolás (network coding) [1] egy olyan megoldás, amely –a csatornakódolással ellentétben- nem a végpontban, hanem az adatok hálózaton belüli csomópontokban keveri (újrakódolja) az adatot. Ezáltal a hálózaton keresztül nagyobb méretű adatfolyamokat lehet átvinni, amely nagyobb hasznos hálózati kihasználtságot eredményez.

Az okostelefonok rohamos fejlődésével egyre nagyobb számítási kapacitás érhető el a mobil csomópontokban, lehetővé téve bonyolultabb számítási műveletek végrehajtását. Annak érdekében, hogy nagyobb hálózati kihasználtságot érjünk el, erőforrásigényes műveleteket kell végrehajtani a kódolni kívánt adaton. A közeljövőben piacra kerülő csúcstelefonok már képesek lesznek ezeket a műveleteket elvégezni minimális késleltetéssel, ezáltal elérve a nagyobb hálózati kihasználtságot. Ez a lehetőség motivált minket a dolgozatban bemutatott munka elvégzésére. Az általunk célul kitűzött feladat egy elosztott hálózati kommunikáció megvalósítása hálózati kódolás segítségével.

A szakirodalom már ismer vezeték nélküli hálózati kódolás alapján megvalósított vezeték nélküli kommunikációt, de ez jellemzően multicast [2] vagy WiFi hálózatok [3] teljesítményjavítására vonatkozott. A mi szándékunk élő videótartalom elosztott kiszolgálása, amelynek a logikai szervezését peer-to-peer alapon is meg lehet oldani. Ezen feladaton belül mi egy megfelelő hálózati kódolási megoldás kiválasztására, illetve annak a jelenleg is elérhető Android alapú okostelefonokkal [4] kialakítható környezetében való vizsgálatára vállalkoztunk.

2 Elosztott hálózati kommunikáció mobil eszközökön

Az elosztott multimédia továbbítás megvalósításával sokan foglalkoztak már korábban. Ezen a szerteágazó kutatási témán belül mi egy jól meghatározott területet vizsgáltunk, amelyben két friss technológiát (hálózati kódolás és Android alapú mobil eszközök) ötvöztünk. Ezen felül a feltételezzük, hogy az eszközök elosztott módon vannak hálózatba szervezve – habár vizsgálataink erre nem terjednek ki, a dolgozatunkban megadunk egy javaslatot, amelyenek részletes kidolgozása és megvalósítása biztosítaná az ilyen jellegű működést.

Ebben a fejezetben a fent érintett három kérdéskört vezetjük be. Áttekintjük a dolgozatunk hátterét, azokat a fogalmakat, technológiákat és szakirodalomban korábban már ismertetett eredményeket, amelyek megalapozták a munkánkat. Ismertetjük a feladatként kiválasztott elosztott hálózat jellemzőit, bemutatjuk a megvalósítandó hálózati csomópontokon futó Android operációs rendszert, valamint a köztük kialakítandó kapcsolatokat, végül pedig ismertetjük a hálózati kódolás alapjait.

2.1 Elosztott hálózat csomópontjai összekapcsolása

2.1.1 Fizikai kapcsolatok

Napjainkban a mobil eszközök számos csomag alapú hálózati kommunikációt megvalósító rádiós interfésszel vannak felszerelve. A nagy hatótávolságú cella alapú rádiós interfészen kívül (pl. GSM, 3G/WCDMA) fel vannak szerelve rövid hatótávolságú (pl. Bluetooth) illetve közepes hatótávolságú (pl. IEEE 802.11) rádiós interfésszel.

Dolgozatunkban ezek közül a 802.11 standard [5] szerint megvalósított hálózati kommunikációs interfészt használtuk, mert alaklmas eszközök közötti közepes hatótávolságú közvetlen és hozzáférési ponton keresztüli közvetett kapcsolódásra is.

Ennek megvalósítására két lehetőség van jelengel:

- Hagyományos WiFi
- WiFi Direct

Hagyományos WiFi

A hagyományos WiFi eszközök hátránya, hogy korlátozott az erőforrás kihasználtsága, ugyanis mindenképpen szükség van egy hozzáférési pontra (Access Point - AP) a hálózatban. Emiatt a hálózatban résztvevő AP egy szűk keresztmetszetet jelenthet, mivel az üzenetek a hálózatban rajta keresztül jutnak el a célhoz. Kellően nagy számú eszközt kapcsolva a hálózatra több

hozzáférési pont szükséges annak érdekében, hogy minél kevesebb késleltetéssel juttassuk el csomagjainkat a hálózatban a megfelelő csomóponthoz.
Az Android operációs rendszerrel felszerelt mobil eszközök képesek hozzáférési pontként működni a hálózatban, viszont ha dinamikusan változó hálózati topológiát szeretnénk megvalósítani, akkor rengeteg plusz számítási kapacitást jelent a folyamatos hozzáférési pont váltás, ami egyrészt lassítja a hálózati kapcsolatot, illetve rossz erőforrás kihasználtásghoz vezet.

WiFi Direct

A Wifi Direct, eredeti nevén WiFi P2P, lehetőséget biztosít WiFivel rendelkező eszközöknek, hogy egymással kapcsolatot hozzanak létre könnyen és gyorsan, anélkül, hogy akármilyen hozzáférési pontra (Access Point – AP) lenne szükség. Ennek segítségével az eszközök gyors kapcsolatot tudnak fenntartani, ami hasznos azoknál az alkalmazásoknál, ahol adatokat osztunk meg másokkal, mint például multiplayer játékok, vagy video streaming [6].

A WiFi Direct lényegében tartalmaz egy szoftver hozzáférési pontot (Soft AP), amit beágyaz az eszközökbe. Amikor egy eszköz belép a WiFi Direct tartományába, csatlakozhat hozzá ad-hoc protokollt használva, majd a kapcsolat létrehozásához kezdeti információkat gyűjt be. A Soft AP bonyolultsága függ a megvalósítandó szereptől. Egy digitális képkocka átküldéséhez elegendő egy egyszerű szolgáltatás megvalósítása, amely csak a kamera használatát igényli. Azonban ha különböző adatokat szeretnénk megosztani interneten keresztül, lényegesen bonyolultabb AP-ra van szükségünk. A WiFi Direct-tel rendelkező eszközök képesek egy-egy vagy egy-több kapcsolatot kialakítani, amely során nem minden a hálózatban résztvevő eszköznek kell rendelkeznie a szabvánnyal. Egy WiFi Direct képes eszköz képes csatlakozni hagyományos WiFi-vel rendelkező eszközhöz is.

2.1.2 Logikai kapcsolat

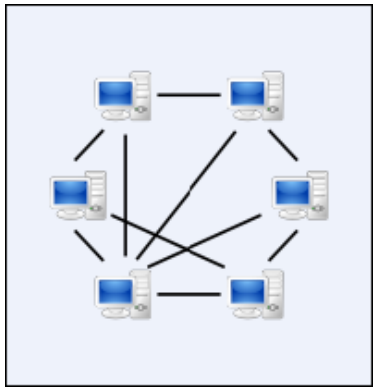
A fizikai réteg fölé egy megfelelő logikai réteget kell helyezni, amely vezérli a csomópontok közötti kapcsolatot. Egy hálózati topológia a számítógép-hálózatok esetén a hálózathoz tartozó csomópontok közötti kapcsolatokat határozza meg.
A hálózati topológia esetében a logikai kapcsolatok szintjén csak a csomópontok közötti összeköttetések a fontosak: sem az összeköttetés módja, sem az átviteli sebessége, sem a csomópontok távolsága, sem az összeköttetés fizikai módja, stb.

A munkánk során egy decentralizált, részben szövevényes (mesh) hálózati topológia megvalósítására törekedtünk. Egy ilyen topológia jól modellezi a gyakorlatban kialakuló elosztott, mobil eszközökből álló hálózatot. A szövevényes topológia azt jelenti, hogy egy csomópont egyidejűleg több csomóponttal is össze van kapcsolva. A mi elképzeléseink szerint az eszközök hálózatba szervezése is így történik. E hálózati modellt tekinthetjük a peer-to-peer hálózati modell alapjának.

2.1.3 Peer-to-peer kapcsolatszervezés

A peer-to-peer (P2P, egyenrangú) paradigma [7] lényege, hogy a hálózat végpontjai közvetlenül egymással kommunikálnak, központi kitüntetett csomópont nélkül. A peer-to-peer fogalom két hasonló, de célját tekintve mégis eltérő fogalomkört is takar: a számítógépek egyenrangú technológiai szintű kapcsolódási módját egy helyi hálózaton, valamint valamilyen célból közvetlenül kapcsolódó szoftver megoldások működési elvét.

A közvetlen kapcsolat hibatűrőbb felépítést, skálázhatóságot jelent. Hátrányai: a nehezebb adminisztráció, az erőforrások pazarló használata, a nehezebb megvalósíthatóság. Egy peer-to-peer típusú hálózat logikai kapcsolatait láthatjuk az 1. ábrán.



1. ábra – P2P hálózati topológia (forrás [3])

A P2P hálózatokban a felhasználók (peerek) erőforrást biztosítanak a hálózat többi peerjei számára, amelyek magukba foglalják a sávszélességet, tárhelyet és a számítási teljesítményt. Amikor egy új csomópont érkezik, és egyúttal az igények száma is növekszik a rendszerben, egyúttal a hálózat teljes kapacitását is növeli. Ezzel ellentétben egy tipikus kliens-szerver architektúrában a kliensek csak a saját igényeikkel törődnek, az erőforrásaikat nem osztják meg.

Ebben az esetben amint több és több ügyfél csatlakozik a rendszerhez, kevesebb erőforrás fog rendelkezésre állni [8].

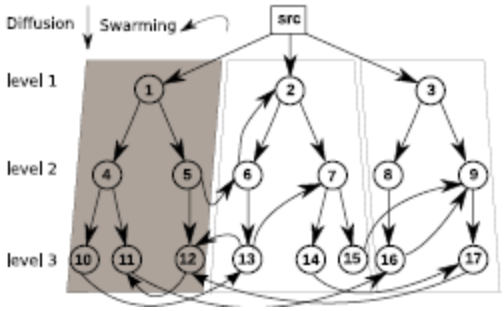
Média átvitele esetén egy kliens-szerver architektúrájú hálózatban a szerver csak korlátozott számú felhasználót tud kiszolgálni saját kapacitásának megfelelően. Ezt a korlátot meghaladva vagy rohamosan romlik a közvetítés minősége, vagy új kliens már nem tud csatlakozni a hálózathoz. Ezzel szemben a peer-to-peer technológia segítségével gyakorlatilag korlátlan számú felhasználó szolgálható ki úgy, hogy a stream minősége is megfelelő marad. Ezt kihasználva egyre több olyan alkalmazást hoznak forgalomba, amely képes a felhasználókat egy közös hálózatba szervezni, ezzel növelve a hálózat kapacitását [9].

2.1.5 P2P streaming

A „streaming média" valós idejű adatfolyam, általában tömörített multimédiás információ Interneten keresztül való kézbesítése, amely a bináris számítógépes fájlformátumokhoz (például mpg) viszonyítva kevésbé célozza meg a videó tartalom teljes hűségű visszaállítását, elsősorban az azonnaliságra összpontosít [10].

Az élő streaming, pontosabban a videó és műsorszórás élő, valós idejű („real-time") közvetítését jelenti interneten keresztül. Ez a folyamat magába foglalhat egy kamerát a videó rögzítésére, egy kódolót a tartalom digitalizálására, egy videó megosztót, ahol a tartalom elérhető a felhasználók számára, illetve egy tartalomközlő hálózatot, mely lebonyolítja az adatforgalmat. Ezeken kívül szükségünk van még kódoló-dekódoló modulokra is, ezek szolgáltatják a megfelelő tömörítő/kibontó algoritmust a kódoló eszközök és lejátszók számára. A szervereken és a lejátszókon ugyanarra a protokollra van szükség a „streaming" adatok továbbítására [11].

A p2p streaming elnevezés a peer-to-peer szoftver-alkalmazásra utal, amely lehetővé teszi a videós tartalmak újraelosztását valós időben a P2P hálózaton. Ezek a videós tartalmak legtöbbször TV csatornák a világ minden tájáról, de egyéb forrásokból is származhatnak [12].



2.ábra – P2P streaming hálózati topológia

Minden felhasználó, amikor letölt egy videót, egyidejűleg lehetővé teszi más felhasználók számára, hogy tőle letölthessék. Ezáltal hozzájárul a szerver foglaltságának csökkentéséhez, ugyanis a tőle letöltő peer már nem kell a szerverhez kapcsolódjon. A 2. ábrán láthatjuk egy ilyen hálózat topológiáját. A videó letöltésének állapota általában mindig késleltetett a forrásvideóhoz képest. A csatornák videó minősége attól függ, hogy hány ember nézi éppen. Minél több felhasználó nézi egyidejűleg, annál jobb minőségben lehet a videót lejátszani [12]. A p2p streaming felépítése hasonlít a BitTorrent fájlcserélőjére [13], azonban streaming esetén kezelni kell a valós idejű kapcsolatból eredő nehézségeket [14]. Ha a felhasználó úgy dönt, hogy egy bizonyos csatornát szeretne nézni, akkor a p2p streaming alkalmazás kapcsolatba lép a keresésért felelős komponenssel, hogy azoknak a peereknek elérhetőségét megtudja, akik éppen megosztják ezt a csatornát. Ezt követően pedig kapcsolatba lép velük, hogy megkapja az adatokat. Az előbb említett komponens figyeli a letöltő peereket, és elérhetővé teszi címüket azon felhasználók számára, amelyek ugyanazt a csatornát óhajtják nézni.

A p2p streaming megoldások hátránya, hogy a média továbbítása/elosztása szorosan összefonódik a p2p hálózatba szervezéssel. Az elérhető megoldások a média továbbítását unicast módon, jellemzően UDP vagy RTP szállítás-rétegbeli protokollok segítségével oldják meg [14]. Ezzel szemben mi azt szeretnénk elérni, hogy a hálózati kódolás nyújtotta előnyöket is kihasználjuk, ezáltal nemcsak a hálózat szélei, a felhasználók eszközei vegyenek részt a megoldásban, hanem a hálózat is. A mi vizsgálataink során a hálózat „belsejében” levő csomópontok is felhasználói eszközök által vannak megvalósítva. Ez egyrészt megnyitja számunkra az utat a hálózaton belül telepítendő megoldások alkalmazására, azok vizsgálatára, miközben a tervezés és megvalósítás során megmaradhatunk a hálózat szélén levő

csomópontoknál, illetve az ott alkalmazott technológiáknál (pl. Android). Másrészt továbbra is alkalmazhatjuk a p2p paradigmát, ami számunkra a médiaelosztásban résztvevő minden csomópont egy logikai átfedő hálózatba (overlay) szervezhetőségét jelenti. A hagyományos IP hálózat belső csomópontjait megváltoztatni, például a hálózati kódolás megvalósításához szükséges modulok telepítését elérni egy hosszabb folyamat. Ugyanakkor a dolgozatban is vizsgált javaslatok és elért eredmények szükségesek ahhoz, hogy középtávon megvalósítható legyen ez a módosítás. A fenti okok miatt a továbbiakban eltekintünk a hagyományosan a hálózaton belüli technológiák vizsgálatával, a megoldásaink során a felhasználói eszközökben általunk alalmazott technológiákra fókuszálunk.

2.2 A hálózat csomópontjai: Android alapú okostelefonok

Az Android egy Linux alapú operációs rendszer [4], elsősorban érintőképernyős mobil eszközök számára. 2007-ben mutatták be először, és 2008 októberében adták el az első Android alapú telefont [4][15]. Azóta a legelterjedtebbé vált a piacon elérhető, okostelefonokon futtatott operációs rendszerek között. Az Android platform egyik nagy előnye a nagyszámú elérhető alkalmazás és azok könnyű telepíthetősége, emiatt nagy számú fejlesztő közössége létezik a világon.

2.2.1 Android platform

Az Android platform réteges felépítésű. A legalsó réteg tartalmazza a Linux kernelt, melynek feladata a memória kezelése, a folyamatok ütemezése és teljesítmény-kezelés. Emellett itt találhatóak az eszközkezelők programjai A Linux kernel réteg fölé épülnek a programkönyvtárak, amelyek használják a kernel szolgáltatásait. Ezek jellemzően C/C++ nyelven megírt könyvtárak és közvetlenül a Linux kernelen futnak.

Részben erre épül az Android futtatókörnyeze, ami egyrészt tartalmazza a Java alapvető könyvtárait, illetve a kulcsfontosságú virtuális gépet, amely lehetővé teszi, hogy az alkalmazások külön példányban tudjanak futni, egymástól elkülönülve külön Linux folyamatként.

E fölött már csak Java forráskódú réteget találunk. Az Application Framework tartalmazza a felhasználó felületet felépítő API-kat, míg a legfelső, Application réteg pedig beépített alkalmazásokat tartalmaz [4][15].

2.2.1 Android alapú fejlesztés

Az Android alapú fejlesztés a Java szabályait követi, de a mobil környezet korlátai miatt van néhány sajátossága. A következőkben röviden áttekintjük az Android alapú fejlesztés legfontosabb elemeit, amelyeket az angol terminológia szerint sorolunk fel.

Activity

Az Android-os fejlesztés legfontosabb komponense. Feladata a felhasználói felület létrehozása és megjelenítése, ami megjelenik az alkalmazás futásakor. Minden egyes ilyen képernyő egy külön-külön Activity. Az alkalmazás tervezésekor fontos figyelembe venni az Activity életciklusát, ugyanis ezen keresztül valósíthatók meg bizonyos interakciók kezelése (például ha a telefont elfordítják, vagy a felhasználó bezárja az alkalmazást). Egyszerre több Activity is futhat, és ezek egy stack-en (verem) kapnak helyet. Ennek a lényege az, hogy ha egy Activity háttérbe kerül, akkor bekerül ebbe a verembe. Ezután akárhány Activity kerülhet rá. A kivétel a berakás fordított sorrendjében történik, tehát az utoljára berakott Activity kerül ki a veremből először.

SharedPreferences

Az alkalmazás által generált adatok tárolására alkalmas. A tárolás kulcs-érték párokban történik. Ezáltal lehetőség van fontosabb változók, vagy beállítások perzisztens mentésére az alkalmazás futása közben, és természetesen ezek visszaállítására is. Az adatok tárolásának legegyszerűbb módja, viszont nagy mennyiségű adat tárolására nem ajánlatos, az Android-ban van, külön erre a célra kitalált SQLite adatbázis.

Intent

Az Intent, vagyis szándék tulajdonképpen egy komponensek közötti üzenet valaminek a végrehajtására, például egy új Activity indítására. Az igazán nagy előnye, hogy a szándékon kívül különféle adatok átadása is lehetséges ennek segítségével az Activity-k közt. Az adatokat sokféleképpen átadhatjuk, de minden formában ez kulcs-érték párokban történik. Ezek a párok viszont csak primitív típusok lehetnek, tehát saját osztály-objektum átadása nem lehetséges.

Content Provider

Olyan komponens, melynek segítségével Intent-eken keresztül az alkalmazás bizonyos részei más alkalmazásokból is elérhetők. Ezáltal nincs szükség több alkalmazásban az azonos funkciókat megvalósító részeket többször megírni, hanem elég egyszer, és Content Provider alkalmazásával ezt a funkciót kiajánlani más alkalmazások számára. Például ha egy alkalmazás

képet akar készíteni, akkor lehetőség van éppen emiatt a gyári kamera alkalmazás segítségét igénybe venni.

Felületi elemek

Minden ilyen elem ősosztálya a View. A legfontosabb tulajdonságokat innen öröklik, amelyek megjelenítés szempontjából fontosak, viszont mindegyik rendelkezik az arra jellemző funkcionalitással.

ListView: Listák megjelenítésére használható. Görgethető és az egyes elemek kiválasztására is lehetőség van.

TextView: Egyszerűbb szövegek megjelenítésére szolgál. A szövegre kattintani is lehet, illetve mérete és színe is tetszőlegesen változtatható.

EditText: Szöveg bevitelére alkalmas mező.

Button: Gomb, amelyre felirat helyezhető, illetve megnyomásakor valamilyen akció hajtódhat végre.

Szálkezelés

Egy Android alkalmazás indításakor létrejön egy processz, amiben a program minden komponense fut. Ezen a processzen belül fut a program fő szála, más nevén: main-thread, vagy ui-thread. Ezen a szálon belül jön létre a program összes komponense.

Gyakran előfordulhat, hogy nagy számításigényű műveletet akarunk végrehajtani. Ilyenkor érdemes ezt egy külön szálban végrehajtani, hogy ne blokkolja a ui-thread végrehajtását. Saját szálakat létrehozhatunk a Thread, vagy Runnable osztályok segítségével.

A szálak képesek üzeneteket küldeni és fogadni. Üzenetek fogadásához regisztrálni kell a szálhoz egy MessageQueue-t, erre való a Looper osztály. A *prepare* metódusával lehet létrehozni egy üzenetsort, majd a *loop* segítségével lehet rajta iterálni. Az üzeneteket a beérkezés sorrendjében, szekvenciálisan képes feldolgozni.

Az egyes üzenetek kezeléséhez egy Handler objektumra van szükség, amellyel üzeneteket tudunk küldeni, illetve feldolgozni. A Handler egy példányát egy MessageQueue-val lehet összetársítani, aminek az üzeneteit fel tudja dolgozni.

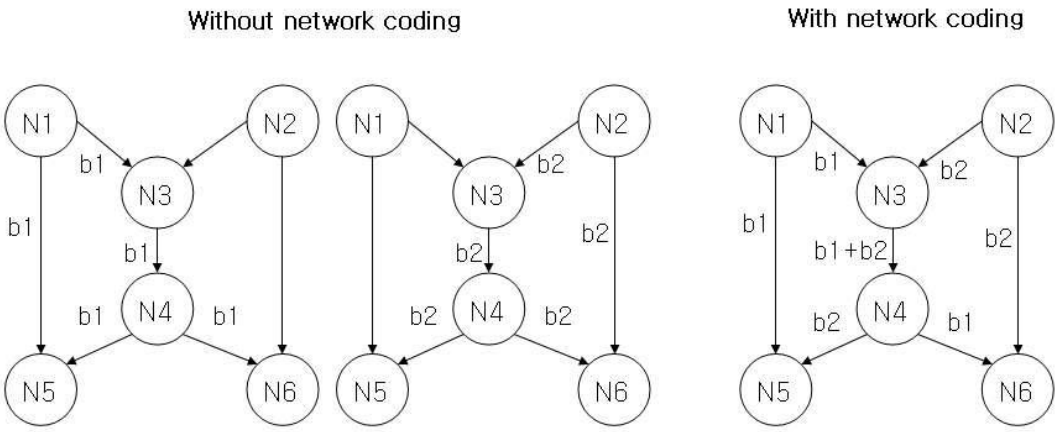
2.3 Network Coding

Amint a Bevezetőben is említettük, dolgozatunkban a hálózati kódolás mechanizmusára építünk. Ebben a fejezetben áttekintjük a hálózati kódolás alapjait, valamint egy olyan speciális hálózati kódolási eljárást, amely különösen alkalmas elosztott hálózatok támogatására.

A hálózati kódolás alkalmazásával a köztes csomópontok nem csak továbbítják a beérkező csomagokat, hanem műveleteket is végrehajtnak rajtuk, pl. a beérkező csomagok lineáris kombinációját képezve. Ezzel multicast alapú kommunikációban a csomagok összekódolása által a hálózati szűk keresztmetszeteken kevesebb adatmennyiséget kell továbbítani, így elérve a magasabb áteresztőképességet [2][16].

Mivel egy streaming media átviteli hálózat hatékonyságát erősen meghatározza a késleltetés, ez lehet egy módja annak, hogy elosztott heterogén hálózatokban több-több pontos kapcsolatú kommunikációt megvalósítsunk anélkül, hogy a kliens/szerver architektúrára támaszkodnánk.

A hálózat kódolás előnyét gyakran a pillangó topológiával magyarázzák (lásd 3. ábra) [1]. Lényegében a sugárzott csomagokat egymással párhuzamosan, egy időben továbbítjuk. A bal oldalon látjuk, hogy a két csomag mely linkeken utazik. A csomagok a hálózatkódolást (Network Coding - NC) alkalmazva úgy kódolódnak, hogy a végpontban, vagy egy ahhoz közeli csomópontban egyértelműen visszaállítható legyen az eredeti adat. A részletes kódolási eljárásokról később lesz szó. Ezzel a módszerrel növelhető a hálózat kapacitása, hiszen ahelyett, hogy a példánál maradva b1 és b2 csomagok külön továbbítódnának, ehelyett a kettő által meghatározott kódolt csomag egy linken utazik {N2;N3}, ami ebben az esetben 50%-al kisebb sávszélesség igényt eredményez. A végpont(ok)ban azért dekódolható az adat, mert b1 és b2 közül az egyiket megkapva a kódolt csomag visszafejthető.



3. ábra - Pillangó topológia NC nélkül, illetve annak használatával [2]

Mindennek tudatában a hálózatkódolás alkalmazásával tehát az információ eljuthat egy adott vevő számára eredeti formájában a forrástól, de lehetséges, hogy kódolva, ekkor a vevőnek dekódolnia kell. A köztes csomópontok képesek továbbítani az információt, de ha úgy kívánja a hálózat, akkor a különböző beérkező folyamokat kombinálva/kódolva egy csomagként küldik tovább a következő csomópontnak. Erre a műveletre erre specializálódott pontokon kerül sor annak érdekében, hogy később az egyéb résztvevők képesek legyenek dekódolni, azaz visszanyerni az eredeti információt.

A hálózat kódolás egy meghatározott működés, viselkedés alapján dolgozik tehát, ahol az előzőekben definiáltak szerint a kódolás/dekódolás során az események egy adott sorrendben követik egymást, így biztosítva a kódolhatóságot és a visszaalakíthatóságot. Ez épp úgy előnye, mint hátránya a technológiának, hiszen határt szab a résztvevők számának függvényében a csomópontok struktúrájának a hálózat egészére nézve. További érzékeny pontja lehet az eljárásnak a kódolás és dekódolás műveletek ideje, ami késleltetést okozhat, ezzel további követelményeket szabva, például a csomagok szinkronizálása.

2.3.1 Véletlen lineáris kódolás

Az eddigiekkel ellentétben létezik egy alternatíva, amely nem ilyen "szigorú" kódolás, hanem véletlenszerű viselkedés alapján működik. A kódoló csomópontok tetszőlegesen különböző együtthatókat rendelnek a csomagokhoz, melyek dekódolhatóságának valószínűsége a

csomópont helyétől is függ. Ebben az esetben minden csomópont független és álvéletlen kódolási együtthatókat használ, így nincsen szükség – ellenben az előzőekkel – a hálózat többi részének ismeretére. A véletlen lineáris hálózatkódolással (Random Linear Network Coding - RLNC) a közbenső, kódoló csomópontok a bejövő üzenetek egy lineáris kombinációját képezik, amit aztán a kimenő élek mindegyikét felhasználva továbbítanak. Ez a kombináció egymástól függetlenül és véletlenszerűen választja ki az együtthatókat egy véges test alatt (Galois-elmélet). Ezáltal a fogadott csomagok nagyon nagy valószínűséggel dekódolhatók lesznek a végpontokon, nyelőkben. Mivel az együtthatók a Galois-test által meghatározott véges tartományból kerülnek ki, ez azt jelenti, hogy a bejövő üzenetek lineáris kombinációjának elemei nagy valószínűséggel lineárisan függetlenek, amit a dekódoló a Gauss-elimináció segítségével elegendő gyorsasággal tud megállapítani, így rekonstruálva az eredeti csomagot [17].

Az RLNC elsősorban a hálózatnak olyan szegmensében használható, ahol több folyam is áthalad. Egy másik megközelítésben létezik az eljárás ugyanazon folyamon belüli csomagokra is [3]. Ez a folyamon belüli hálózatkódolás hatékony megoldás a hibákból adódó csomagvesztések hatásai ellen, ami a veszteséges vezeték nélküli linkeken különösen fontos, és nagy jelentőséggel bír.

Az NC működése több követelményt támaszt a hálózattal szemben. Ilyen például az erőforrás, sávszélesség, számítási sebesség, szinkronizált adatátvitel. Ezek mindegyike a más-más szerepekben levő csomópontoknál különbözik.

- Az adó, azaz a csomagokat létrehozó forrás az átküldendő adatot multicast módon szórja. Ehhez nem szükséges nagy számítási igény, ez a tartalomtól függő szokásos erőforrásokat igényel, elegendően nagy feltöltési sebességgel, nem túl nagy számítási sebességet.

A köztes csomópontok működhetnek kódolóként, valamint "futárként".

- Amennyiben az utóbbi "futárként" működik, akkor a bejövő csomagokat a kimenő interfészén továbbítja mindenféle módosítás nélkül. Ez megint csak átlagos tulajdonságokkal rendelkező csomópont, mivel különösebb feladata nincsen, így csak a hálózat fennmaradásához szükséges követelményeket kell teljesítenie.

- A kódoló a hálózatkódolás legfontosabb résztvevője. A fentiek alapján a kódoló csomópont a kimenő link áteresztőképességét és kapacitását egyaránt növelni tudja, ez a technológia lényege. Mind vezetékes, mind vezeték nélküli hálózatban jól alkalmazható. Látványos eredményeket azonban az utóbbiban lehet vele elérni. Mint fentebb szó volt róla, többféle kódolási eljárás létezik. A bitenkénti XOR műveletet használó folyamat már minimális számítással eléri a kívánt hatást. Ez viszont szűkebb keresztmetszetet ad a csomagok számának tekintetében. Az RLNC, azaz a véletlenszerű együtthatókkal a bejövő csomagok lineáris kombinációját képző kódolás már rugalmasabb, és a randomizálás (álvéletlen szám generálásával) még mindig nem túl nagy számítási sebességet megkövetelő művelet. Az erőforrásigény abban mutatkozik meg, hogy a kódolt és "kódolatlanul" utazó csomagok szinkronban maradjanak, azaz a vevő egyértelműen, elég gyorsan ki tudja számolni a dekódolt üzenetet - utóbbi esetben inverz mátrixot képezve, ezzel lehetővé téve az információ összeállítását. Így tehát a gyorsaságon sok múlik, vagy ezen áldozva a szinkronizálást kell elvégezni adott idő után. Jelen dolgozatban ismertetett mérések ezzel a kérdéssel is foglalkoznak. Sávszélesség-igénye kisebb, mint egy átlagos p2p hálózati csomópontnak, hiszen jóval kevesebb adatot küld, mint amennyit fogad.
- A végpontban levő, vevő csomópontok végzik a dekódolást. (Gauss-elimináció, XOR művelet, stb.) Ez tehát a számításhoz szükséges erőforrások nagyságára érzékeny. Emellett fontos művelet a több folyamból érkező csomagok sorba rendezése, válogatása.

A hálózati kódolási megoldásokat eddig jellemzően unicast, nem valósidejű adattovábbításra [18], multicast átvitel támogatására [2] és szenzor adat átlagolására (data averaging) [19] javasolták. A mi javaslatunk a valós idejű média streaming továbbítására fókuszál, amelyek manapság jellemzően nem használnak multicast megoldást. Az előbb hivatkozott megoldásokhoz viszonyítva a mi javaslatunk az adattovábbítás és a multicast megoldások „között” helyezkedik el. Ennek az új felhasználási módnak a megvalósításhoz szükséges tervezési döntéseket, a megvalósítás menetét a következő fejezetben ismertetjük.

3 Hálózati kódolás alapú elosztott kommunikáció

A fentebb javasolt hálózati kódolási eljárás (RLNC) alkalmazhatóságát élő multimédiatartalom elosztására egy Android alapú okostelefonokból kiépített teszthálózatban vizsgáltuk. Ebben a fejezetben áttekintjük az kódolás megvalósítását, a teszthálózat felépítését és a videóátvitel biztosításához szükséges megoldásainkat.

3.1 P2P hálózat

Elosztott vezetéknélküli hálózatunkban az adattovábbítás vezérlését peer-to-peer (p2p) mechanizmusokat felhasználva javasoljuk megoldani. Ugyanakkor a mi esetünkben magát az adat továbbítást már nem a fájlcserélőkben megszokott módon végezzük (mint például a BitTorrent [13]), hanem a hálózati kódolást figyelembe véve, folyamatosan küldünk a forrástól a célnak.

Egy p2p átfedőt (overlay) kell létrehoznunk és fenntartanunk, hogy a vezérlési adatokat el tudjuk juttatni a résztvevőknek. Áttekintettük a széles körben elterjedt megoldásokat, keresve a lehetőséget, hogy azokat alkalmazva oldjuk meg a feladatot. Az AllJoyn [22] egy nehézkes, komplex keretrendszer, amelyik ráadásul nem is érett technológia, sok apró hibája van. A JXTA [23] keretrendszer nagy publicitást kapott, de támogatása megszűnt és a programozási felülete (API) is túl bonyolult a mi feladatainkhoz. A harmadik megoldás a Sip2Peer [21] protokoll, amely a SIP komplexitását hozza magával.

A címzés a p2p overlayben eltér az IP hálózati címzéstől, minden peernek (csomópontnak) egyedi kulcsa van, amelyet a kommunikáció során, a fizikai és hálózati kapcsolataitól függetlenül, végig megőriz. Ez a megoldás segít elrejteni a csomópontok mobilitását is. A peer kulcsa és IP címe közti nyilvántartását a p2p rendszernek kell megoldania.

A mi javaslatunk egy virtuális kétirányú gyűrű, melyben minden peer legalább két szomszédjával van kapcsolatban (megj. ez csak a vezérlési adatok továbbítása szempontjából releváns, az adatcsomagokat nem ez a p2p rendszer routolja). A gyűrű alapú p2p megoldások esetében a szakirodalom leghivatkozottabb megoldása a Chord [20]. Ebben egy hash függvénnyel oldják meg az IP címek és peer azonosítók közti átjárást. A mi esetünkben ettől eltérünk, mivel a hálózatunk mérete sokkal kisebb egy világméretű p2p hálózatnál – a teszthálózatunkban 100 forrásnál nem lesz több csomópont. Ezáltal a keresés egyszerűbben történhet, a gyűrűn haladva a szomszédok szomszédjait kérdezve.

A megoldás robosztusságát úgy növelhetjük, ha egy peer több szomszéddal van kapcsolatban. A gyűrű fenntartásához elég a két szomszéd ismerete, de ha szeretnénk növelni a robosztusságot – és a gyűrű szimmetriáját fenntartani, úgy tudjuk r-szeresen növelni a redundanciát, hogy a két irányban r előre-, illetve hátramutató mutatót tárolunk. A peerek felfűzése a gyűrűre az azonosítójuk sorrendjében történik, ami lehet az első belépéskor viselt IP cím, és az azonosítót a Chord szabályai szerint oszthatjuk ki.

Ha egy adott peer 2r szomszédot ismer és N peer van a rendszerben, akkor a legrosszabb esetben N/2r üzenet kell egy sikeres kereséshez. Ha r a rendszer redundanciája, ami ha egy elég alacsony szám (3) és megtartjuk az előbbi N=100 felső korlátot, akkor egy keresés legfeljebb 17 üzenetet generál, ami a p2p hálózatok esetében, ahol sok vezérlési üzenetet használnak, egy vállalható többletterhelés, különösen mivel az adattovábbítás során nem kell vezérlési üzeneteket cserélni (mint pl. a BitTorrent letöltés alatt a csonkok folyamatos lekérésekor igen).

Az első csomópont belépésekor egy tagú lesz a gyűrű, minden mutató ugyanerre a címre mutat. Amint sorra belépnek az új csomópontok, úgy kell a mutatókat frissíteni, a Chord szabályai szerint. A belépéshez elég egy bootstrap csomópontot ismerni, vagy egy jól ismert címet dinamikusan átirányítani az éppen aktuális, változó bootstrap peerre, esetleg egy multicast címen lehet ezt a csomópontot hirdetni.

Amint egy csomópont belépett a gyűrűbe, bármely másik peerre képes keresni, vele kapcsolatot létesíteni és neki üzenetet küldeni az azonosítója alapján. Ha csak az IP címét ismeri, akkor egy gyűrűn kívüli, külön üzenettel érheti el az IP routing szabályai szerint, elkérheti az azonosítóját és később bármikor, a Chord szabályai alapján tud vele kapcsolatba lépni.

A megoldásunk előnye az egyszerűsége és a robosztussága. Továbbá, mivel nem kell folyamatosan tároljuk a rendelkezésre állását egy peernek (ráér azt a kapcsolatfelvételkor megtudni), a gyűrűnek alacsony a fenntartási költsége, ami előnyös egy dinamikus környezetben.

3.2 Véletlenszerű hálózati kódolás megvalósítása

Az implementált alklamazás a következőkben leírtak alapján működik.

A csomagtovábbításoz UDP vagy TCP protokollt kell használnia. Annak érdekében, hogy csökkentsük a szállítási protokoll overhead-ét, az UDP-n való csomagtovábbítás mellett döntöttünk.

A továbbítani kívánt stream előállításához a kamera által generált előnézeti képeket használtuk fel. A kódoló folyamat ezekből a képekből előállítja a megfelelő frame-ket, amelyek ezután

továbbítódnak a hálózat csomópontjaihoz. E folyamat előnye, hogy könnyebben detektálható a kódolás-dekódolás következtében előforduló hiba, ugyanis azonosítani tudjuk, hogy melyik frame volt a hibás a médiafolyamban.
A csomópontok a kapott frame-eket szerepüktől függően vagy továbbítják azokat módosítás nélkül, vagy kódolják azokat az RLNC fejezetben leírtak alapján.
Végül a 'cél' csomópont dekódolja a kapott csomagokat, szintén a leírtak alapján, majd megjeleníti az összeállított képkockákat.

Struktúra:

1. BlockLevelExample.java: random network coding hálózat használatának példa megvalósítása.
2. CodingVectorLevelExample.java: ncutil könyvtár használatának példa megvalósítása. Ezt a két példát használja fel a MyPreview.java illetve az Utils.java
3. MyPreview.java: vezérli többek között a kamerát. Kódolatlan csomagok létrehozása a kamera segítségével. Az Utils.java végrehajtja a kódolást. Ezek lineáris kombinációjából létrehozza az elküldendő kódolt csomagokat. A végpontban a dekódoló segítségével visszaállítja az eredeti csomagokat.
4. Utils.java: a létrehozott csomagok kódolásának végrehajtása itt történik meg. A kódolatlan csomagokat transzformálja megfelelő kódolt formátumra. Ehhez különböző bitműveleteket hajt végre.
5. MainActivity.java: az alkalmazás vezérlése itt történik meg.

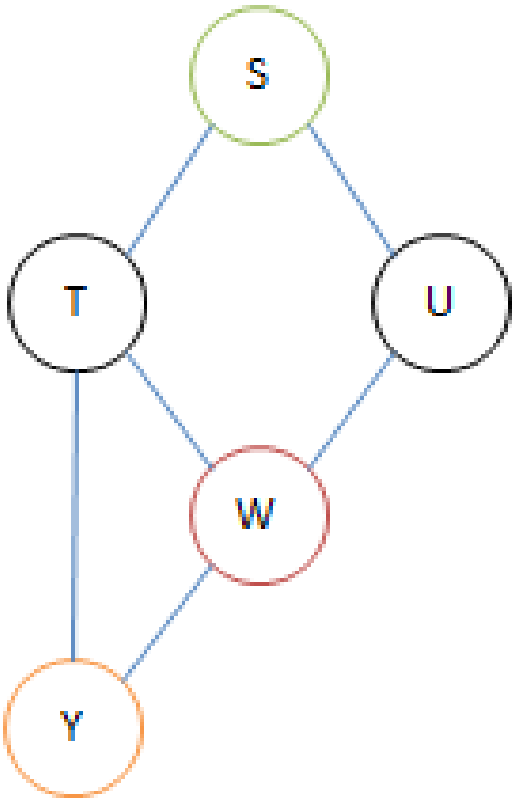
A csomópontok broadcast üzenetekkel derítik fel a hálózatban szereplő egyéb résztvevőket, így biztosítva a lehetőséget a szükséges kapcsolatok kialakításához.

3.3 Teszthálózat

A mérések során a következő eszközök álltak rendelkezésünkre:

- 1 db HTC Desire [24][25]
- 4 db Google Nexus S [26]
- 1 db Google Nexus 4 [27]
- 1 db Sony Ericsson Ray [28]

A fenti felsorolásból az első három készüléket az Android fejlesztője referencia eszközként forgalmazta a fejlesztők számára, mivel a piacra kerülésük időpontjában azok az adott időszakban elérhető legjobb hardvert biztosítja.



4. ábra - Teszt hálózat felépítése

Ezt kihasználva a teszthálózatunkban gyakorlatilag az elmúlt három generációs okostelefonokon vizsgálhattuk a megvalósításunk alkalmazhatóságát. A SonyEricsson Ray készülék egy középkategóriás készülék volt megjelenése évében, a jelenlegi készülékek között egy átlagos, belépő szinttel megegyező teljesítményre képes. Mivel gyakorlatilag bármelyik multimédia képes készüléket képvisel, vizsgálataink során majd a multimédia forrásaként használjuk, ennek a választásnak a további részleteit a következő fejezet elején tárgyaljuk.

A csomópontok az 4. ábrán ismertetett hálózati struktúra szerint épültek fel.

Szerepek a hálózatban:

- S: Forrás
- T,U: Egyszerű továbbítást végeznek, megjelenítik a kapott médiafolyamot
- W: Kódoló NC alapján

- Y: Dekódoló, visszaállítja az eredeti csomagokat, majd megjeleníti a médiafolyamot

Mint látható, négy különböző funkciót ellátó szereplő alkotja a teszthálózatot, melyek lefedik a fentebb ismertetett NC hálózat minden funkcióját. A forrás (S) feladata a médiafolyam előállítása, annak továbbküldése U és T számára. A médiafolyam képkockánként felváltva kerül a két csomóponthoz, azaz a páratlan sorszámúak a T, a párosak az U részére továbbítódnak. Ők a megfelelő csomópontokhoz kódolás nélkül továbbítják a kapott csomagokat a p2p rendszerekhez hasonlóan. Emellett meg is jelenítik a médiafolyamot. A kódoló (W) feladata a csomagkódolás, jelen esetben a véletlenszerű lineáris kódolás (RLNC) végrehajtása, majd a kódolt csomagok továbbküldése a dekódoló (Y) számára. A kódolónál nem történik videó megjelenítés. A dekódoló feladata a kódolatlan illetve kódolt csomagokból, az eredeti csomagok visszaállítása, majd a médiafolyam megjelenítése.

Látható, hogy a legfontosabb szerepet a forrás, kódoló illetve dekódoló látja el. A forrás az előállított kép minőségében játszik fontos szerepet. Egy HD-képes telefon HD minőségű képet fog előállítani, illetve továbbküldeni, amit egy nem HD-képes telefon nem fog ugyanúgy megjeleníteni. Előzetes várakozásaink szerint ez a mérések során is jól látható lesz.

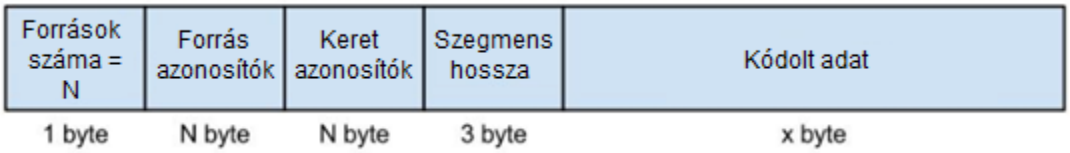
3.4 Mobil videó

Mivel az Android operációs rendszert folyamatosan fejlesztik és készülékek között különbségek lehetnek a telepített verzió tekintetében, mi az Android 2.3-as, régóta jelenlevő verzióból indultunk ki, ez mindegyik tesztkészülékünk esetében rendelkezésre állt. Ugyanakkor ez behatárolta az élő videótovábbítási képességeit az eszköznek. Ennek megfelelően nem tudjuk az RTP (Real Time Protocol) kényelmes videótovábbítási többletszolgáltatásait használni. Sajnos az alacsonyszintű (natív) élő videótovábbítási képességei a legújabb Android 4.2-es verzióban sem megfelelő, hiába használható fejlett videókodoló, a bevitt pufferelési késleltetés miatt gyorsabb megoldást kellett fejlesztenünk. A valós idejű megoldásunkhoz UDP protokollt használtunk. A videó forrása a készülék kamerája, amelynek előnézeti képéből (camera preview) kockánként rakjuk össze a videófolyamot. Ezeket a nyers képeket egy képkódoló könyvtár eljárásával kódoljuk, és ezt küldjük UDP csomagként a célhoz. Ez a megoldás annyiban volt előnyös számunkra, hogy képkockánként önálló képünk van, így a kódoló fejlesztése és hibajavítása

során sokkal egyszerűbb volt javítani a megoldásunkat, egy-egy keret önállóan megjeleníthető volt, nem függött más kerettől (képkockától).

3.5 Folyamok és videó keretek azonosítása a közös csatornákban

A kódoló megfelelő transzformációk segítségével (Galois-elmélet és lineáris kombináció [1]) létrehozza a kapott csomagokból a kódolt csomagokat. Ezek erőforrás igényes műveletek, elsősorban a folyamatosság miatt, ami elengedhetetlen az élő műsorszórás esetében. A dekódoló a kapott kódolt illetve kódolatlan csomagokból Gauss-elimináció segítségével visszaállítja az eredeti csomagokat. Ezen művelet végrehajtása szintén erőforrás igényes. Ezen csomópontnál megint több bejövő adat van, amelyek szinkronizálása is szükséges a dekódolással egyidejűleg.



5. ábra – Adatcsomag fejléce

Ehhez nagy segítséget nyújt a továbbított UDP csomagok fejlécének felépítése. A fejléc felépítését az 5. ábrán ismertetjük. Egy adatcsomag összesen N forrás adatát tartalmazhatja, azaz N folyamot kódolunk egy útvonalon (inter-flow network coding). Minden egyes forrásra megadjuk az azonosítóját és a hasznos adat hosszát. Az is lehetséges, hogy egy adott folyamnak két különböző keretét kódoljuk, ezeket azonosítjuk a harmadik mezőben (intra-flow network coding). A mi megvalósításunkban, a teszthálózatunk korlátai miatt az egy bájtos méret bőven elegendő, hiszen a források száma a százas felső korláton belül van. A kódolt adat szegmens (ennek a hosszát kódoljuk 3 bájtban a negyedik mezőben) nemcsak a kódolt adatot, hanem a kódolási együtthatókat is tartalmazza.

4. A megvalósított RLNC megoldás vizsgálata

Vizsgálataink során a fentebb említett három fontos feladatot ellátó szereplő, különböző telefonok által való megvalósítását vizsgáltuk. A T és U csomópontok szerepét permanensen egy-egy Google Nexus S telefon valósította meg. Ezt azért engedhettük meg, mert ezen csomópontok szerepe csak átlagos csomagtovábbítási erőforrást igényel, jelentős változást nem okoz a jobb, illetve rosszabb teljesítményű készülék használata. Ezen kívül két másik csomópontot szintén Google Nexus S eszközök valósítottak meg, míg az ötödik készülék jelentette a változtatást az alap vizsgálati elrendezéshez képest.

4.1 A videóátvitel alkalmazás vizsgálata

Először azt vizsgáltuk meg, hogy milyen hatása van a forrás készüléknek a generált videófolyamra. Mivel alapvetően a hálózati kódolás viselkedésére voltunk kíváncsiak, nem a videótovábbító alkalmazásunkra, ezt a vizsgálatot is a mérési rendszerünk beállítására használtuk. Amint az előző fejezetben is említettük, több készülék állt rendelkezésre, különböző videó képességgel. A méréseink alapján 12 képkocka/másodperc (frame per second - fps), 128kbps folyamot folyamatosan tudott a rendszer szolgáltatni, ami még biztosította ez élő videó élményt. A méréseink alapján (QoE értékelése, folyamatos videó visszajátszás képesség a célállomáson) a megvalósításunk működőképes, képes a szükséges videófolyamot biztosítani a kódolónak.

A későbbi vizsgálataink szempontjából fontos döntés volt, hogy a mérések alapján a Google Nexus S-el hasonló streamet biztosít a Sony Ericsson Ray készülék is, így azt mindig forrásként használtuk. Ezáltal a mérési topológiánkban a többi négy csomópontot ugyanazzal az egy készülékkel lehet megvalósítani, ez lett a referencia összeállításunk. Amint a későbbiekben leírjuk, a további vizsgálatainkban ennek a topológiának egy-egy csomópontját változtattuk, az úgy elért eredmények pedig egységes keretrendszert biztosítottak az értékeléshez.

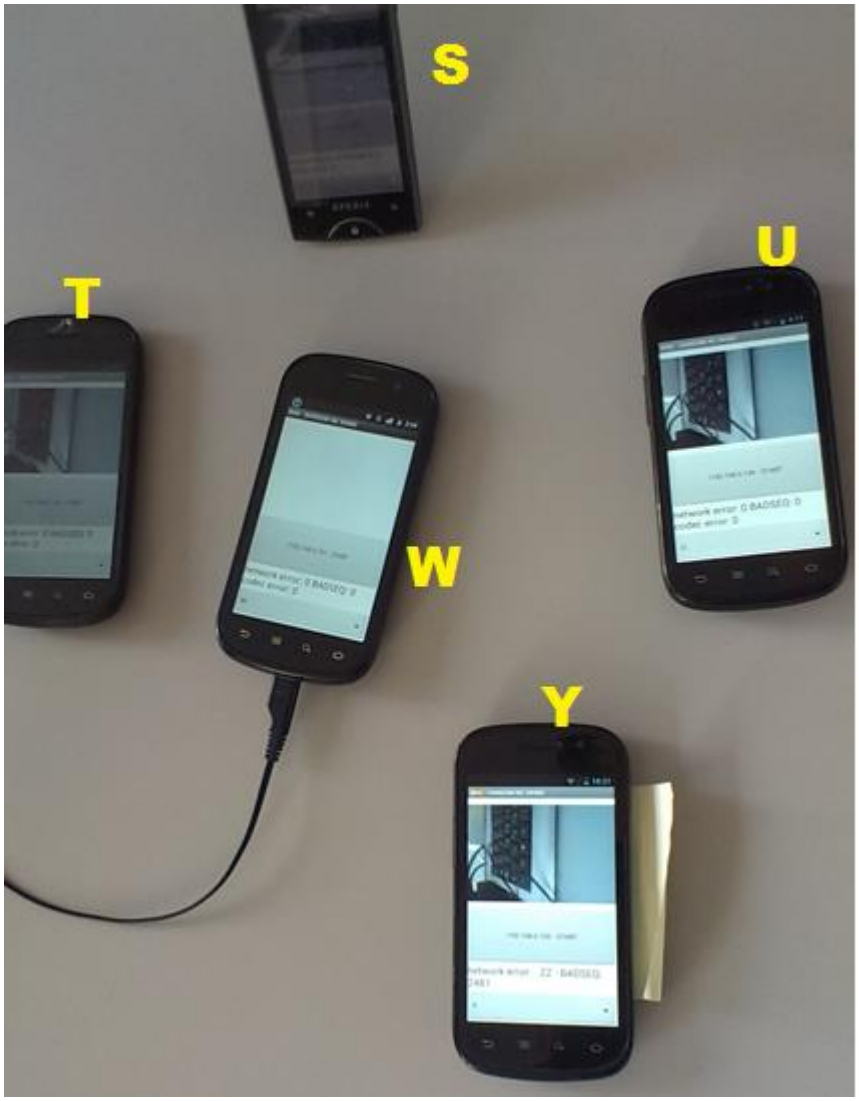
4.2 Mérések – alap elrendezés

Ebben az alfejezetben azt a mérési elrendezést ismertetjük, amit a fent ismertetett előtörténet alapján állítottunk össze és a továbbiakban viszonyítási pontként használtuk. Továbbá röviden bemutatjuk az XOR alapú vizsgálatainkat, mivel az a legegyszerűbb NC megoldásként a mérési rendszerünk validálására is használható. Az XOR alapú sikeres működés volt a minimum feltétel

ahhoz, hogy a bonyolultabb RLNC alapú megvalósítás vizsgálatát egyáltalán értelme legyen elvégezni.

Az videófolyam átvitele mérései során láttuk, hogy a rendszerünk vizsgálatához a Sony Ericsson Ray készülék forrás csomópontkénti használata egy megfelelően alkalmazható, mert az általa generált adatfolyam a gyengébb készülékek által is jól kezelhető, a T és U csomópontok nincsenek túlterhelve.

A 6. ábrán látható, hogy a két köztes, egyszerű továbbítást végző telefonon, illetve a cél telefonon megjelenik a médiafolyam.

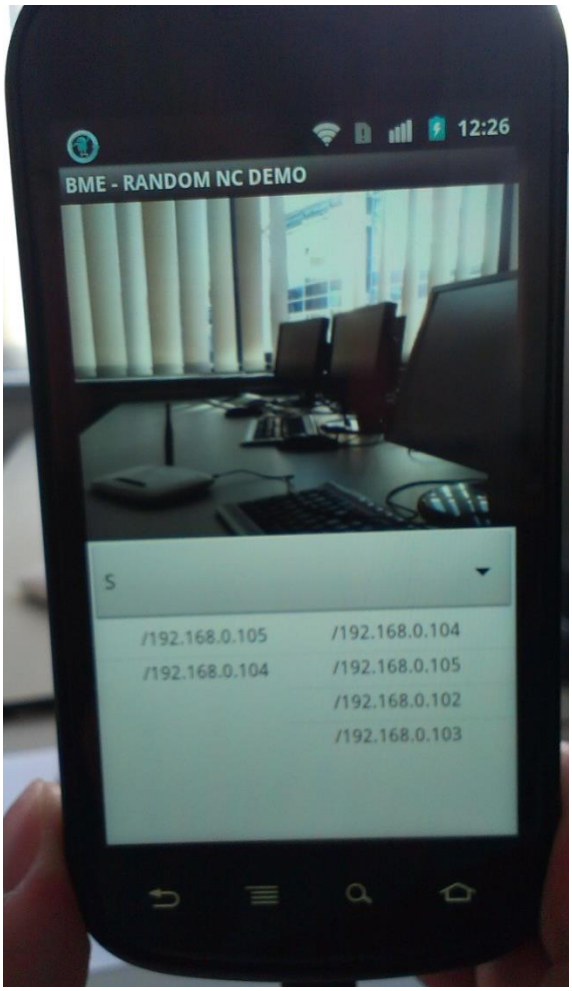


6. ábra - Teszt hálózat működés közben

A bitenkénti XOR megoldást sikeresen meg tudtuk valósítani, a kódolók esetében mindegyik készülék képes volt fenntartani a videófolyamot. Ugyanakkor, a fentiekben részletesettek szerint a videólejátszó alkalmazásunk nem tudta az Android kernel szintű eljárásait használni. Képes volt valós időben megjeleníteni a videófolyamot, de ezt a minőség (Quality of Service - QoS) rovására (pl. alacsonyabb volt a keret-ráta –fps- és felbontás –resolution-). Ezeket a korlátokat az eszközök hardverének és az Android fejlesztői környezetének fejlődésével át lehet lépni. Vizsgálataink alapján az alkalmazhatóságot a videólejátszó oldalon levő eszköz befolyásolja a legjobban, mivel az nemcsak a videófolyamot kell megjelenítse, hanem a dekódolás műveletét is el kell végezze. A QoE-t értékelve elmondhatjuk, hogy a HTC Desire készüléken megfigyelt videó akadozott egyedül, másik két készüléken az alkalmazás képes volt folyamatosan működni. Ez a gyakorlatban azt jelentette, hogy intenzívebb képváltások esetében a HTC Desire célállomás elveszítette a szinkront, a videófolyam élvezhetetlen volt. Összefoglalva az eddigieket, azt állapítottuk meg, hogy az XOR alapú hálózati kódolás esetében a szűk keresztmetszet az eszköz multimédia képessége, és nem a kódolás/dekódolás folyamata.

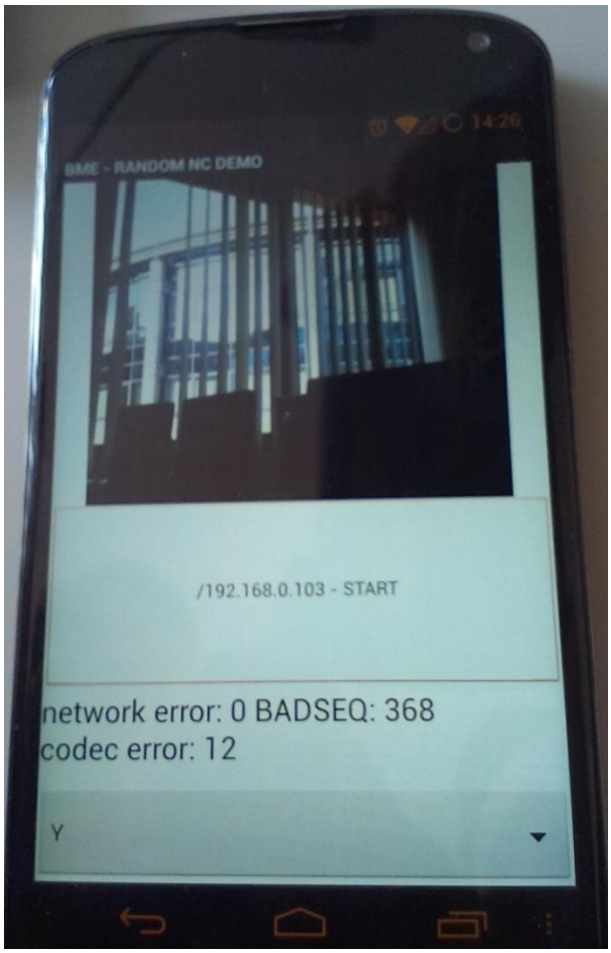
4.3 RLNC mérés

A következő képen (7. ábra) a forrás csomópont adatai láthatóak, rajta maga a továbbküldött médiafolyam, a számára látható csomópontok (jobb oldali lista), illetve a szomszédos csomópontok elérhetőségei (bal oldali lista). Pontosabban, tesztalkalmazásunkban a szomszédos csomópontok a „downstream” kapcsolatok végpontjait jelölik, azaz azokat a csomópontokat, akiknek az alkalmazásnak el kell küldenie az adatot.



7. ábra - Forrás (S) kezelő felülete az elérhető csomópontok listájával

Az alábbi képen (8. ábra) a dekódoló szerepet betöltő telefon látható a folyamat közben. Az egyes hibatípusok, azok aktuális értékeivel illetve a médiafolyam látható a kijelzőn.



8. ábra - Vevő (Y) kezelőfelülete a hibaszámlálókkal

A mérések során a következő adatokat, eredményeket rögzítettük, hasonlítottuk össze:

- *network error*: A csomag fejlécében hibás adat szerepel. Ez akkor fordulhat elő, ha a vezetéknélküli továbbítás során bithiba lép fel, így a fejléc használhatatlan marad és az egész csomagot el kell dobni.
- *codec error*: a dekódolás sikeres, de a megjelenítés nem lehetséges, azaz a dekódoló előállított egy keretet, de azt a megjelenítést végző dekóder nem tudja képkockaként értelmezni. Ez akkor fordulhat elő, ha a hasznos adatként átküldött csomagrész (kódolt adatrész az 5. ábrán) sérült. Ebben a két esetben a hálózat alsó két rétegében fellépő hibákat számoljuk, azaz a vezetéki közeget jellemzi. Ezeket a hibákat alacsonyabb rétegű kódolással lehet javítani (pl. CRC alkalmazása a fejlécben, kódolási koefficiensek védelme CRC kódokkal).

- *badseq error*: hibás szegmens, azaz nem használható egy adott videókeret. Pl. elcsúszott a szinkronizáció, a megkapott csomagok alapján nem sikeres a Gauss-elimináció, nem végezhető el a dekódolás. Amíg az előző két esetet a hálózat alsó két rétegében fellépő véletlen hibákat jelzi, addig ezt az eset az összes olyan hibákat összefoglalja, amelyek sikeres vezetéknélküli átvitel ellenére akadályozzák a sikeres videólejátszást. Ez azt jelenti, hogy akkor kapunk „badseq error”-t, ha a hálózatban fellépő hibákat az eljárásunk nem tudja megoldani. Tulajdonképpen ez az a hiba, ami akkor jelentkezik, amikor a teszthálózatunk teljesítménye nem elég ahhoz, hogy valós időben feldolgozza a videófolyamot.
- *quality of experience* (QoE): A lejátszott média minőségének összehasonlítása a forrás (S), illetve a vevő (Y) által megjelenített képek alapján.
- *Médialejátszás megszakadása*: Az időintervallum, ameddig egy elkezdett egyenletes, megfelelő minőségű médiafolyamot a vevő (Y) le tud játszani. Amennyiben sok hibát generál a rendszer, a videókockák visszaállítása a kódolt adásból már több ideje nem lehetséges. Az 5. ábrán ismertetett fejléc információk segítségével egy rövid kihagyással a videófolyam megjelenítését az alkalmazás tudja folytatni, de ez az esemény egy könnyen mérhető erős QoE romlás indikátora.

4.4 Eredmények

A mérések során a hálózat csomópontjai Google Nexus S [26] készülékek voltak, amelyek számítási kapacitása megfelel egy 2013-ban forgalmazott közepes képességű okostelefonénak. A mérésünk során a kódoló egységet változtattuk. Ez a mostanáig piacra már korábban bevezetett, az Android operációs rendszert kifejlesztő Google cég által referenciaként forgalmazott három generációt képviselő eszközöket. A Google Nexus One [25] készülék a legrégebbi készülék (esetünkben azzal a hardverben megegyező HTC Desire készüléket [24] használtuk), míg a Nexus 4 készülék [27] forgalmazását ebben a negyedévben állították le, de a következő generációs készülék még nem elérhető a piacon – azaz a legjobb minőséget képviselik. A nexus S a köztes választékot képviselik.

A mérés során a teszthálózatunkban a YouTube videómegosztó portálon korábban felső korlátként meghatározott 10 percig figyeltük az élő adást. Mindhárom beállításnál létrejött a videókapcsolat, a videófolyam észlelése során (Quality of Experience) nem akadozott a videó.
A hálózat minőségét jelző *network error* és *codec error* értékek minden esetben nagyon kis értékek voltak, 20-30 között. Ennek alapján elmondhatjuk, hogy a méréseinket egy jó környezetben végeztük el.

Ugyanakkor mi figyeltük azokat az eseményeket, amikor a képkockákat nem sikerült megjeleníteni. Ennek értékeléshez egy saját mérőszámot használtunk, amely a dekódolás sikertelenségét, valamint a kódolási késleltetésből adó szinkron-eltolódást együttesen értékeli. Ez utóbbi akkor történik, ha az egyenes ágon beérkező csomag megelőzi a kódolt csomagot. Ez a hibajelzés akkor is előjön, ha a vezeték nélküli továbbítás miatt történt hiba. Mivel az eljárásunknak ezt az eseményt is el kell fednie, a mérőszám ezt is tartalmazza.

Az 1. táblázat foglalja össze az eredményeket, amelyeket 10 mérés átlagolásából értük el. Látható, hogy a Nexus One készülék is képes a kódolási feladattal megbirkózni, de 25%-al nagyobb hibával, mint a legfejlettebb eszköz. Azonban mindhárom esetben egy idő után, a videófolyam szinkronhibái oda vezettek, hogy megszakadt a videófolyam – ezt az időpontot tartalmazza az 1. táblázat harmadik oszlopa. Ebben az esetben zavaróbb a gyenge eszköz hátránya – az alkalmazásunkban ezt a videófolyam újraindítása áthidalja.

1. Táblázat – Mérési eredmények hatperces médiafolyam továbbításakor (a kódoló eszköz függvényében)

Kódoló eszköz	Összes hiba	Első megszakítás időpontja
nexus one (htc desire)	3500	4.05 perc
nexus S	3300	5.30 perc
nexus 4	2800	6.29 perc

Azt is megvizsgáltuk, hogy milyen hatással van a megoldásunkra a dekódoló állomás számítási kapacitásának változása. Ebben az esetben a kódolást egy Nexus S készülék végezte. A mérés eredményeit a 2. táblázatban foglaltuk össze. Az első észrevételünk az volt, hogy a leggyengébb

készülékkel gyakorlatilag nem tudtunk folyamatos videófolyamot megjeleníteni, emiatt nem is mértünk értékelhető hibás eseményt.

2. Táblázat – Mérési eredmények hatperces médiafolyam továbbításakor (a dekódoló eszköz függvényében)

Kódoló eszköz	Összes hiba
nexus one (htc desire)	értékelhetetlen
nexus S	3300
nexus 4	2000

A vizsgálatainkat összefoglalva elmondhatjuk, hogy sikerült létrehoznunk egy olyan megoldást, amely hálózati kódolás alapján végzi az élő multimédia folyamok elosztását a napjainkban elérhető eszközökön. A megoldásban a legnagyobb terhelésnek a dekódoló egység van kitéve, de már több éve elérhető mobil eszközök számítási kapacitása is elég ahhoz, hogy folyamatos lejátszást biztosítson. Szintén fontos megállapítás, hogy a hálózati kódolás számításigénye nem ró elviselhetetlen terheket a ma elérhető mobil eszközökre, azaz további kutatások során az RLNC technológia alkalmazható peer-to-peer élő médiaelosztás támogatására.

5 Összefoglalás

Ebben a dolgozatban azt vizsgáltuk, hogy miként támogatható egy elosztott hálózati kommunikáció a hálózati kódolási eljárással. A munkánk során felvázoltuk egy ilyen hálózattal szembeni elvárásokat, kiválasztottuk a feladatra megfelelő hálózati kódolási eljárást, terveztünk egy protokollt, ami képes az elosztott tartalmakat kezelni.

Android okostelefonokon megvalósítottuk az élő videó tartalmat elosztó alkalmazást, valamint a hálózati kódolási eljárást. Különböző generációs okostelefonokkal vizsgáltuk a javaslatunk alkalmazhatóságát. Megállapítottuk, hogy a jelenleg elterjedt eszközökön már valós idejű médiatartalom szórását lehet megvalósítani ezzel az eljárással.

A továbbiakban a hálózati csomópontok logikai szervezését, egy peer-to-peer keretrendszert tervezünk megvalósítani, hogy a teszthálózatunkban dinamikus környezetben vizsgálhassuk a javaslatunkat.

Irodalomjegyzék

[1] R. Koetter, M. Medard, “An algebraic approach to network coding”, IEEE Trans. on Networking, October 2003

[2] D. Traskov, Lenz, J., Ratnakar, N. and Médard, M., “Asynchronous Network Coded Multicast” accepted to the 2010 ICC Communication Theory Symposium

[3] S. Katti, D. Katabi, W. Hu, H. Rahul, and M. Medard, “The importance of being opportunistic: Practical network coding for wireless environments”, in Proc. 43rd Annual Allerton Conference on Communication, Control, and Computing, 2005

[4] Ekler P., Fehér M., Forstner B., Kelényi I., "Android-alapú szoftverfejlesztés", ISBN 978-963-9863-27-9, Szak Kiadó, 2012, elérhető: http://szak.hu/android-fejl/

[5] IEEE Standard for Information technology – „Telecommunications and information exchange between systems Local and metropolitan area networks, Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications”, 2012, forrás: http://standards.ieee.org/getieee802/download/802.11-2012.pdf

[6] WiFi Direct Alliance honlapja - http://www.wi-fi.org/discover-and-learn/wi-fi-direct

[7] Fox, Geoffrey. "Peer-to-peer networks." Computing in Science & Engineering3.3 (2001): 75-77.

[8] Elena Digor, „Kademlia Measurements”, 2009, forrás: http://cnds.eecs.jacobs-university.de/courses/nds-2009/digor-report.pdf

[9] Xiaojun Hei, Chao Liang, Jian Liang, Yong Liu and Keith W. Ross – „A Measurement Study of a Large-Scale P2P IPTV System”, IEEE Transactions on Multimedia, Vol. 9, 2007, pp. 1672 – 1687

[10] Adobe Dynamic Media Group – „A Streaming Media Primer”, 2000, forrás: http://www.adobe.com/products/aftereffects/pdfs/AdobeStr.pdf

[11] Shahzad Ali, Anket Mathur and Hui Zhang, „Measurement of Commercial Peer-To-Peer Live Video Streaming”, forrás: http://www.cs.cmu.edu/~hzhang/papers/PPStreamingMeasurementWRAIPS06.pdf

[12] Nazanin Magharei, Reza Rejaie, „ISP-Friendly Live P2P Streaming”, forrás: http://mirage.cs.uoregon.edu/pub/tr09-07.pdf

[13] J.A. Pouwelse, P. Garbacki, D.H.J. Epema, H.J. Sips – „The bittorrent P2P file-sharing system: measurements and analysis”, Lecture Notes in Computer Science, 2005, Vol. 3640, pp. 205-216.

[14] SopCast honlapja - http://sopcast.org/

[15] Google Android fejlesztői dokumentáció – available online from: http://developer.android.com/guide/components/index.html

[16] „Efficient and Simple Multicasting”, HIPERCOM projekt beszámoló, forrás: http://hipercom.thomasclausen.net/resteam/core/index.php?mod=projects&name=40a2db02a16aa9406a775809075c0b57, letöltés dátuma: 2013.10.22.

[17] Gajic B, Riihijrvi J, Mhnen P., “Performance evaluation of network coding: Effects of topology and network traffic for linear and xor coding”, Journal of Communication, vol. 4(11), pp. 885-893, 2009

[18] Gkantsidis, Christos, and Pablo Rodriguez Rodriguez. "Network coding for large scale content distribution." In Proc. of IEEE INFOCOM 2005. Vol. 4. IEEE, 2005

[19] X. Zhang, G. Neglia, J. Kurose, ”Network Coding in Disruption Tolerant Networks, Network Coding: Fundamentals and Applications” Elsevier Science (Ed.) 2011

[20] Stoica, I.; Morris, R.; Karger, D.; Kaashoek, M. F.; Balakrishnan, H., "Chord: A scalable peer-to-peer lookup service for internet applications", 2001, forrás : http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

[21] SIP2peer honlapja - https://code.google.com/p/sip2peer/

[22] AllJoyn Androidra vonatkozó honlapja - https://www.alljoyn.org/docs-and-downloads/android

[23] Sun Microsystems Inc. – „JXTA v2.0 Protocols Specification”, forrás: https://jxta.kenai.com/Specifications/JXTAProtocols2_0.pdf

[24] HTC Desire készülék honlapja - http://www.htc.com/www/help/htc-desire/

[25] Google Nexus készülékek honlapja – http://www.google.com/nexus/

[26] Google Nexus készülék S honlapja - http://www.android.com/devices/detail/nexus-s

[27] Google Nexus 4 készülék honlapja - http://www.google.com/nexus/4/

[28] Sony mobil készülékek honlapja – http://www.sonymobile.com/global-en/products/phones