Hur hanterar *Discord* växande datamängder?

Samuel Andersson
Lunds Tekniska Högskola
Lund, Sweden
sa3855an-s@student.lu.se

André Frisk
Lunds Tekniska Högskola
Lund, Sweden
an8218fr-s@student.lu.se

Axel Tobieson Rova
Lunds Tekniska Högskola
Lund, Sweden
ax2070to-s@student.lu.se

***Abstract*—VoIP är ett sätt att kommunicera via Internet med text- och röstchatt. Discord är ett VoIP-program som primärt används inom gamingvärlden. Denna applikation har undersökts med hjälp av paketsniffaren Wireshark där datapaket från röst- och textchatt undersöktes. Vidare undersöktes hur applikationen fungerar under stress. Hur påverkas trafiken beroende på hög belastning av skickade meddelanden? Finns det särskilda protokoll för att hantera detta? Hur påverkas trafiken beroende på antal aktiva i samtalet? Vad för komplexitet verkar programmen för dataöverföring uppvisa? Det visade sig att Discord hanterar trafik på olika sätt beroende på hur många som befinner sig i samtalet. Vid testet med hög belasting av meddelanden (spam) använde sig Discord av en annan förbindelse för att skicka användarna dit istället.**

I. Introduktion

Idag används VoIP, Voice over Internet Protocol, allt mer och populariteten växer. VoIP används inom till exempel Skype som många har hört talats om. VoIP är ett sätt att kommunicera via Internet med text- och röstchatt. Skype har länge varit ledande inom VoIP och år 2019 hade Skype 300 miljoner aktiva användare [1]. Dess tillväxt har dock hamnat i baksätet till förmån för andra aktörer, villiga att implementera funktionaliteter som Skype saknar [2].

År 2015 kom en ny konkurrent till marknaden vid namnet Discord. Discord är ett VoIP-program som släpptes av företaget Hammer & Chisel med målsättningen att skapa en plattform där individer kan kommunicera med vänner runt om i världen som spelar datorspel, samt få det lättare att prata med människor de bryr sig om [3]. I början var Discord inte populärt då det saknade funktioner såsom videosamtal som många andra VoIP-program hade, vilket gjorde att marknaden fortsatt dominerades av Skype. Idag har Discord lagt till fler funktioner såsom skärmdelning och videosamtal och har mer än 100 miljoner aktiva användare varje månad [3]. Då Discord är gjort främst för kommunikation i sammanhang med datorspel är det särskilt viktigt att programmet kräver så lite processorkraft som möjligt samt försöker vara latensfritt, då processorkraften borde gå till spelet medan kommunikationen fungerar snabbt och smidigt.

Säkerhet är en viktig funktion för VoIP då mycket delas och pratas om på servrarna. På Discord måste man acceptera en annan användare som vän för att de ska få skicka meddelanden privat eller att vara med i samma community vilket gör att inte vem som helst kan kontakta vem de behagar [3]. Konversationer är även privata då data inte samlas in för intäktsgenererande ändamål eller för att skapa användarprofiler, utan enbart för användarupplevelse och prestanda [4]. Då satsningen för användarsäkerhet är viktig för Discord får även vem som helst försöka hitta säkerhetsrisker i programmet [5]. En avgörande egenskap för applikationer som riktar sig mot en stor användarbas är förmågan att hantera växande datamängder.

A. Frågeställning

Rapportens syfte är att under detta tema undersöka hur Discord hanterar specifika frågor kring text- och röstchatt. Alltså hur de klarar av specifika situationer som kan uppstå vid användning av VoIP generellt. Frågeställningarna som ska besvaras är därmed följande:

- Hur påverkas trafiken beroende på hög belastning av skickade meddelanden? Finns det särskilda protokoll för att hantera detta?
- Hur påverkas trafiken beroende på antal aktiva i samtalet? Vad för komplexitet verkar programmen för dataöverföring uppvisa?

Dessa frågeställningar besvaras med hjälp av teori från kurslitteratur och nätet samt med paketsniffaren WireShark.

II. Teori

A. Position av datacentral

IP-adresserna som förekom vid undersökningen var likartade och därför togs adresserna som förekom ofta som Discords servrar. Datan som skickas från Discord via detta experiment gick via USA där vi med hjälp av IP2Location.com fick ut positionen på servern samt vilken ISP som tilldelats. Vid inloggning och textkommunikation användes följande:

- 162.159.128.233 - USA, San Francisco, CloudFlare Inc
- 35.186.224.47 - USA, Kansas City, Google LLC

Trafiken går främst via CloudFlare Inc. som är ett CDN (Content Delivery Network) som erbjuder skydd till nätverk och cyberhot samt molntjänster. Vid röstkommunikation användes följande:

- 162.245.205.4 - USA, Los Angeles, i3D.net B.V
- 135.128.137.89 - USA, Atlanta, NCR Corporation

Trafiken går främst via i3D.net som är ett företag som erbjuder global high-performance hostings som ska ge låg latens på nätverket till VoIP som Discord har som krav för sin vision.

B. Client-server

Client-server-modellen är en traditionell modell inom internetanvändning där användaren (client) hämtar information från servern. Detta leder till asymmetrisk trafik där klienten hämtar mer data än vad det skickas till servern. All form av utgående data skickas till servern som behandlar datan och skickar det till rätt destination. Detta medför att datorer som kommunicerar med varandra inte vet mer än den datan som skickats [6]. Detta ger skydd i och med att andra användare inte kan få tillgång till information som inte delats. Client-Server används av Discord, var det finns servrar utspritt i världen som användare kan fördelas mellan beroende på vilka rutter som är bäst. Figur 1 visar ett exempel på ett Client-Server nätverk.

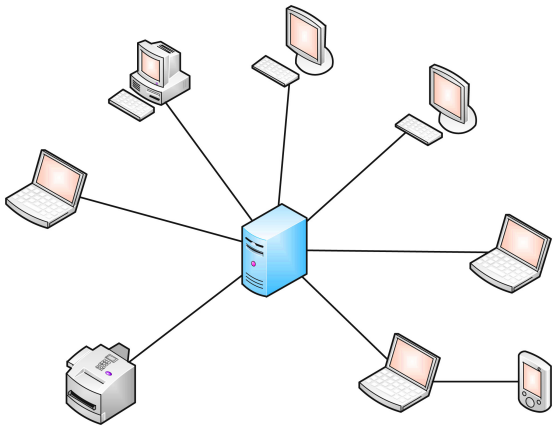

Fig. 1. Exempel på ett Client-server nätverk [10]

C. Komplexitet och multiparty-kommunikation

Discord är ett typiskt exempel på multiparty-kommunikation; system där flera parter kooperativt ska utbyta information med varandra i något syfte. Att undersöka hur man mest effektivt förmedlar data mellan olika agenter är en av informationsteorins fundamentala raisons d'etre. Matematisk formalisering och undersökning av effektivitet är i fokus inom komplexitetsteori.

Studiet av komplexitet utgår från att undersöka tillväxtbeteendet hos en resultantvariabel i fråga, oftast tid eller rum, som funktion av storleken på någon eller några parametrar, för en given algoritm. Då den matematiska teorin främst intresserar sig för de underliggande strukturerna i dess studieobjekt, och generaliserbarheten däremellan, studeras för komplexitetsteorin istället funktionsklasser, där en algoritm faller under en komplexitetsordning O ifall dess tillväxt efter något givet värde på parametern kan begränsas uppåt av någon funktion av ordningen O. Formellt skriver vi

$$f(x) = O(g(x)) \text{ as } x \to \infty$$

om $\exists x_0, M \in \mathbb{R}$, sådant att

$$f(x) \leq Mg(x), x > x_0$$

Då vi ska undersöka optimerad kommunikation mellan flera agenter är därför komplexiteten av intresse; särskilt tillväxtbeteendet för datamängden som behöver överföras beroende på antal användare.

Den historiska komplexitetsanalysen för multiparty-kommunikation utgår till stor del från en simplifierad kommunikationsmodell där två agenter tillsammans ska försöka komplettera input-vektorn till en boolesk-värd funktion, men bara har en input var, introducerad av Andrew Yao 1979 [7] och generaliserad till fler agenter av Chandra et al. 1983 [8]. En väsentlig skillnad mellan dessa modeller och den multiparty-kommunikation vi undersöker är att i den distribuerade uträkningen behöver bara en av agenterna nås av den fullständiga informationen, medan i flerpersonsamtal vill man att alla ska nås. Således inkluderar samtalsmodellen informationsöverföring som hade varit överflödig i det andra fallet, och är således inte öppen för alla dess optimeringar. De kan vara av särskilt intresse som vidare läsning för läsare intresserade av distribuerade system, och ger tillika en fingervisning om vikten av effektiva protokoll och nätverksarkitekturer.

III. Experiment

För undersökningen användes en dator med IPv4 adress på 192.168.1.37 med public IPv4 address 85.30.178.52 kopplad till en router via Wi-Fi med IPv4 192.168.1.0. Nätverkskortet i datorn var av typen Realtek RTL8822BE 802.11ac PCIe Adapter. Undersökningen utfördes även på Lund Tekniska Högskola uppkopplad till Lunds Universitets nätverk eduroam via Wi-Fi till IPv4 10.9.227.186 vars ISP (Internet Service Provider) är Lunds Universitet som går via SUNET som är slutgiltiga ISP. Eduroam använder sig av NAT (Network Address Translation) för att skapa privata IPV4 adresser av de publika IP adresserna.

Innan undersökningarna stängdes datorns internetapplikationer och bakgrundsprocesser av i största mån. Därefter startades WireShark och data började samlas in. För att hitta Discords nättrafik laddades ett program ner från Microsoft vid namn TCPView som synliggör alla processer som använder sig av nätverk till datorn [9]. Med TCPView kunde portarna som används för Discord lokaliseras och applicerades som TCP- och UDP-filter i WireShark.

Totalt utfördes fyra tester. Det första testet ämnade säkerställa vilka protokoll som Discord använder. Det andra undersökte huruvida Discord använder sig av något filter som hanterar spam, medan de sista två undersöker datamängden som brukas vid användning av text- och röstchatt samt hur Discord hanterar privata respektive gruppsamtal.

Det första testet utfördes genom att utifrån informationen i TCPView lägga till portarna till Discord i WireShark som filter så att enbart trafiken från Discord visades. Genom att inspektera paketen som skickades kunde man konstatera att protokollen UDP, TCP och TLSv1.2 används. Dessa protokoll kontrollerades via nätet.

Testet av spamhantering skedde via Wireshark i två moment. Först utsattes en användare för extrema mängder textmedde-

landen under kort tid medan applikationen var i bakgrunden. När applikationen togs upp och alla meddelanden öppnades kom all trafik in på direkten och skapade en väldigt hög ”peak” i mängden paket (se figur 2). Moment två testades genom att även här utsätta en användare för extrema mängder textmeddelanden, men med applikationen uppe. Detta gav de lägre peaks i datatrafiken i I/O Graph som syns i figur 2.

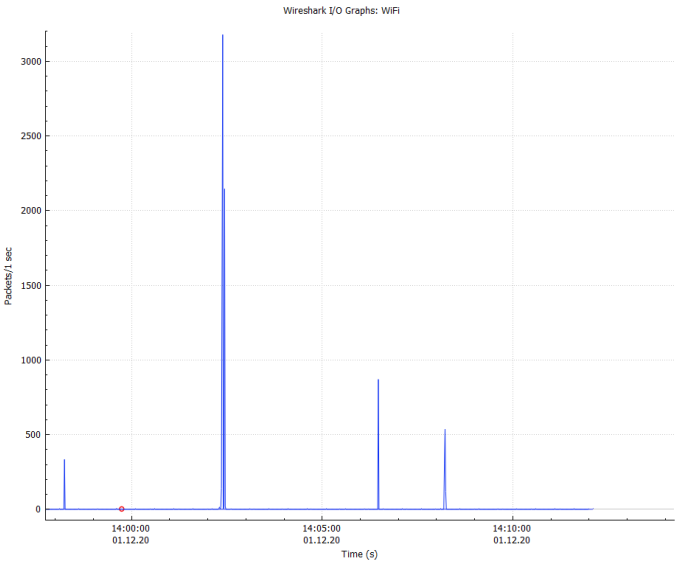

Fig. 2. Bandbredd vid hög belastning av textmeddelanden

Det tredje testet undersökte hur mycket bandbredd som användes vid textchatt med hjälp av WireShark. Mätningen av textchatt utfördes genom att skicka 10 textmeddelanden sammanlagt där ett meddelande skickades var 10:e sekund (se figur 3).

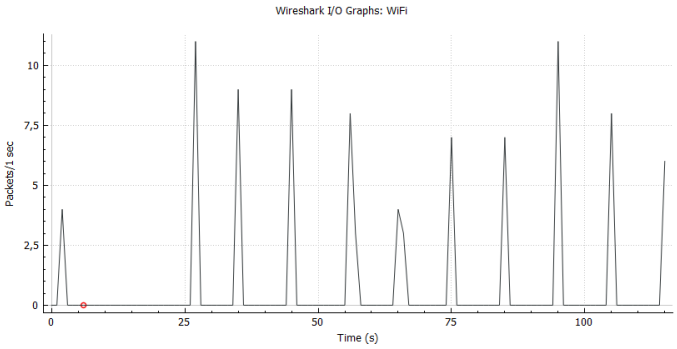

Fig. 3. Bandbredd vid mottagna textmeddelanden

Det fjärde testet undersökte hur Discord hanterar privat- och gruppsamtal med UDP-protokollet samt hur mycket bandbredd som används inom röstchatt. WireShark-mätningen startades och ett privatsamtal startades mellan två personer betecknat av den svarta färgen i grafen i figur 5. Efter en viss tid bjöds det in en till person till chatten och ett gruppsamtal startades som bytte UDP-port, detta visas med den rosa färgen i grafen. En fjärde person bjöds in vid tidpunkten den röda pricken markerar på grafen. Efter detta kunde bandbredden för röstsamtal konstateras.

IV. Resultat

Bandbredden undersöktes för både passivt och aktivt tillstånd. I det passiva tillståndet skickar Discord 2st TCP-paket var 40:e sekund (se figur 4). Detta beror på att Discord skickar en ”keep-alive” förfrågan och försöker ta reda på om förbindelsen används och om den då ska fortfarande användas eller inte.

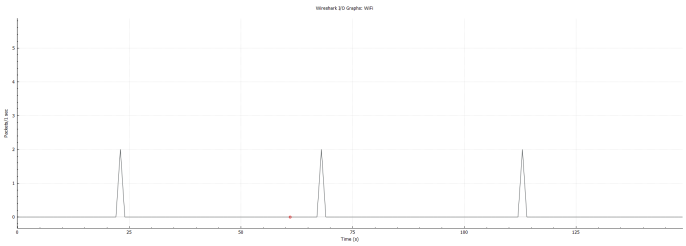

Fig. 4. Bandbredd vid passivt tillstånd

Vid aktivt tillstånd undersöktes bandbredden för skickade meddelanden till privat chatt samt ett öppet röstsamtal. Vid testningen av textchatten där 10 textmeddelanden skickades med 10 sekunders mellanrum motsvarade ett textmeddelande i snitt 7 paket i form av TCP och TLS (se figur 3).

Röstsamtal använder istället UDP-paket, där trafiken vid samtal med en person har ett snitt på ca 100 paket per sekund, medan gruppsamtal som testades med tre och fyra personer gav ett snitt på ca 45 paket per sekund (se figur 5). Det byttes även UDP-port, vilket motsvaras av färgbytet i grafen.

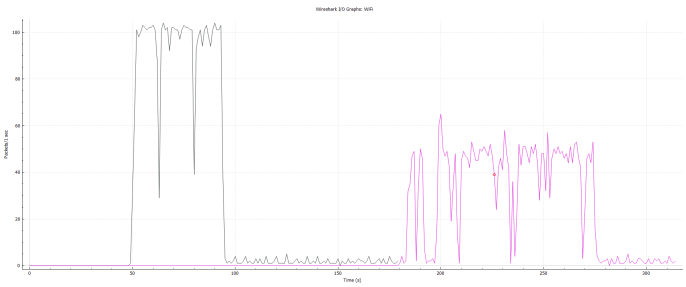

Fig. 5. Bandbredd vid privat- och gruppsamtal

I testet av hög belastning av textmeddelanden (även kallad ”spam”) öppnade Discord en ny TCP-förbindelse mer anpassad för högre volymer. Fönsterstorleken ökade från 500 bytes till 16MB. Servern vi kommunicerade med ändrades från Cloudflare till Google. Krypteringsprotokoll ändrades från TSLv1.2 till TSLv1.3. Discord bytte även tillbaka till tidigare förbindelse när antalet meddelanden återgick till normal frekvens.

V. Diskussion

A. *Spam*

Undersökningen av hur Discord hanterar spam visade sig vara mer intressant än väntat. Under normala dataflöden var

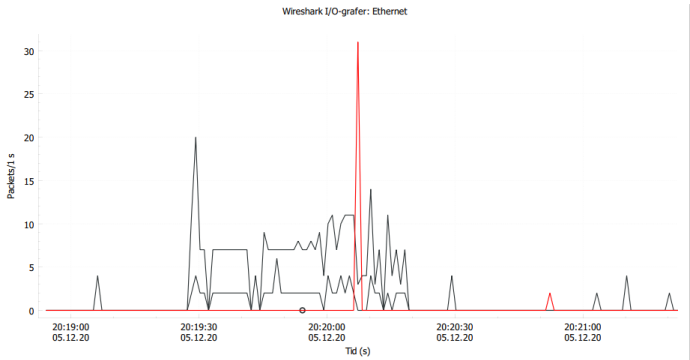

Fig. 6. Exempel på spam. ”Spam-port” har röd färg.

den redan etablerade TCP-förbindelsen statisk. Det som observerades under mätningen var att vid spam öppnades istället en ny port och all data skickades till denna, för att sedan återgå till tidigare port när dataflödet minskade (se figur 6). Först hittades inget om denna teknik och det misstänktes vara en säkerhetsåtgärd från Cloudflare för att förebygga TCP-attacker. Vidare undersökning visade att de första paketen i

Protocol	Length	Info
TCP	66	55816 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
TCP	66	443 → 55816 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1430 SACK_PERM=1 WS=256
TCP	54	55816 → 443 [ACK] Seq=1 Ack=1 Win=131328 Len=0
TLSv1.3	640	Client Hello

Fig. 7. Första paketen på spam-port

denna förbindelse (se figur 7) var ”3 way handshake ”(SYN, SYN-ACK, ACK) där klient och server kom fram till att öka fönsterstorleken för att bättre utnyttja den effektiva bearbetningskapaciteten mottagaren har. Den tidigare förbindelsen var begränsad till fönsterstorleken 500 bytes, det vill säga att maximalt kunna skicka 500 bytes innan ett ACK från servern behövs. Denna nya förbindelse har en fönsterstorlek på drygt 16MB och kan härledas genom att multiplicera ”Win” och ”WS” (se figur 7, första paketet) [11]. Detta ger oss möjlighet att skicka 16MB till serverns buffer, vilket är en avsevärd skillnad från tidigare förbindelse.

Med detta i åtanke stämde ej vår hypotes: förbindelsen öppnas troligtvis inte av säkerhetsskäl utan snarare för att hantera den ökade mängden meddelanden. En smart lösning att temporärt migrera de mest högljudda användarna till en mindre belastad länk med nya spelregler för att kunna leverera data snabbare.

TCP är ett tillförlitligt protokoll som tillhandahåller flera säkerhetsmekanismer som bland annat kvittenser och återsändning vid förlust. Det finns även praktiska skäl till varför TCP-paket generellt är av mindre storlek, till exempel för att undvika IP-fragmentering. Detta gäller framförallt för sämre nätverk där en ”Round-Trip Time” (RTT) tar lång tid. I detta fallet, när stora mängder data ska levereras snabbt på ett bra nätverk är förinställningar inte särskilt gynnsamma. Det skickas många paket och leveransen kommer begränsas av RTT. Därför ändras fönsterstorleken så klienten kan skicka fler paket vid varje tillfälle. På första SYN ses också en förfrågan om SACK PERM = 1 som aktiverar Selective Acknowledgement [12], vilket kort beskrivet underlättar spårningen av specifikt vilka paket som missas eller inom vilket intervall. Detta är nödvändigt då det kommer att skickas så stora mängder utan kontroll, för att undvika att allt behöver skickas på nytt.

Kommunikationen byter också från tidigare IP *[162.159.128.233, CloudFlare] till [35.190.80.1, Google Cloud]* och byter säkerhetsprotokoll från tidigare TLSv1.2 till TLSv1.3. Vi kan bara spekulera kring varför detta görs. En rimlig anledning till varför de byter från Cloudflare till deras back-end Google-server skulle helt enkelt kunna vara att prioriteringarna vid överföring har ändrats. Anledningen till att använda CDN är framförallt att få en server närmre användaren och minska RTT. Men i denna kommunikation försöker de egentligen bara erbjuda så stort fönster som möjligt och som förhoppningsvis lyckas ta emot all trafik direkt utan kommunikation fram och tillbaka, och därför är deras Google Cloud-server troligtvis mer lämplig för detta.

Vi kan inte hitta en anledning till varför TLSv1.3 är mer fördelaktigt i detta scenariot och antar att det är på grund av något praktiskt skäl och inte för ökade hastigheter; kanske att Google Clouds verktyg helt enkelt bara använder denna version, eller att någon av deras spam-verktyg anpassades till denna version.

Något annat som noterades under mätningarna var att förhållandet mellan antalet skickade paket jämfört med hur många som togs emot av mottagare ändrades drastiskt på denna spam-port, från ett tidigare relativt proportionellt samband där 10 paket resulterade i 7 mottagna hos mottagare. Vid spam kunde 2500 skickade paket resultera i 200 paket hos mottagare. Detta är troligtvis på grund av en mer effektiv packetering av meddelande hos servern, det vill säga flera meddelanden per paket. Det saknades möjlighet att öppna och avläsa paketen för att kontrollera om denna hypotes stämmer och därför kommer det endast att nämnas som en observation. Detta kan vara intressant för kommande studier av applikationen att undersöka vidare.

B. Komplexitet och röstsamtal

Då vi endast har tillgång till trafiken över våra egna nät finns det begränsningar för vad för sorts inferenser vi kan göra utifrån den empiriska datan. Det finns även begränsningar för vad för slutsatser vi kan dra på grund av saker vi faktiskt känner till om Discords protokoll. Något vi tidigt observerade i testerna av UDP-trafik under röstsamtal var att trafiken verkade proportionell mot hur mycket det talades. Detta åskådliggörs särskilt tydligt i det mittersta stycket av Figur 4, mellan ca $t \approx 95$ och $t \approx 180$, då alla mikrofoner stängdes av.

Mycket riktigt verifierar även Discord i [13] att de m.h.a röstdetektion i audiodatan kan reducera trafiken under tysta perioder, vilket är en rationell adaption då kostsam trafik inte ska behöva skickas i onödan.

Detta skapar dock vissa komplikationer för att externt analysera den teoretiska komplexiteten, men kan ge oss annan

information om trafiktillväxten i naturliga sammanhang: Låt

$$M = \sum m_i,$$

där m_i är datan från en given individ i, representera den totala mängd taldata som ska kommuniceras mellan de n individer som deltar i samhället vid en given tidpunkt.

Eftersom personer i samtal inte godtyckligt pratar i mun på varandra, kan vi för naturliga sammanhang anta en relativt konstant eller åtminstone väldigt begränsat växande talmängd M för större grupper, under antagandet att att Discords detektionssystem hanterar trivialt "noise" på ett effektivt sätt.

Under client-server-modellen, vilket Discord använder enligt [13], bör en deltagare i då ha en trafikmängd över sitt nät på ungefär

$$(M - m_i) + m_i = M.$$

Med andra ord relativt konstant för naturliga sammanhang.

I Figur 4 övergår vi vid den röda punkten ($t \approx 228$) från 3 till 4 personer i samtalet. Detta är en ökning med 33%. Om trafiken över de enskilda klienterna berodde av antalet deltagare (exempelvis linjärt) borde detta även medföra en proportionerlig ökning av trafiken, men fullt förenligt med hypotesen ovan gör det inte det. Trafiken är snarare närmast konstant. Därav verkar inte klienternas nät behöva belastas mycket mer berorende på antal deltagare, vilket är positivt för användarupplevelsen.

Dock utsätts serversidan naturligtvis för större belastning. Hur detta hanterats överlag har berörts tidigare i stora drag. Vår bästa gissning utifrån den information vi har är att de använder en motsvarighet till tidigare beskrivna naiva modell med normal kryptering (strömchiffret Salsa20 för ljudsamtal [13]), vilket borde motsvara linjär kommunikationskomplexitet ($M \approx$ konstant).

En särskild observation och kanske den mest intressanta från figur 4, är att vid övergång från 2 (första platån) till fler personer i ett samtal så byts UDP-port, och trafiken går ner. Vi har ingen entydig information om vad detta kan bero på, men vår hypotes är att det är en mekanism med liknande syfte som för bytet av TCP-port vid spamhanteringen.

VI. Slutsatser

Discord hanterar växande datamängder på ett effektivt sätt och åstadkommer för röstchatt relativt konstant belastning på användarsidan trots växande deltagarantal vilket gör mycket för användarupplevelsen. Den totala belastningen förväntas bero av detta linjärt. Särskilda protokoll för spamhantering och hantering av fler personer i röstsamtal minskar även belastningen genom att minimera onödig trafik. Detta minimerar den totala kostnaden även om tillväxtordningen förblir linjär. För meddelanden används en annan förbindelse mer lämplig för större datamängder och tillväxtordningen med avseende på textmängd tycks ej vara linjär. Det görs effektivare paketering av meddelanden vid högre volymer.

References

[1] C. Smith, *26 Skype Statistics and Facts (2020)*, Juli 2020, [Online]. Available: https://expandedramblings.com/index.php/skype-statistics/, Accessed: December 2020

[2] T. Warren, *Microsoft's Skype Struggles Have Created a Zoom Moment*, Mars 2020, [Online]. Available: https://www.theverge.com/2020/3/31/21200844/microsoft-skype-zoom-houseparty-coronavirus-pandemic-usage-growth-competition, Accessed: December 2020

[3] Discord, *Vi skapade Discord så att du ska kunna skapa ett sammanhang*, [Online]. Available: https://discord.com/company, Accessed: December 2020

[4] Discord, *Discords engagemang för en säker och trygg upplevelse*, [Online]. Available: https://discord.com/safety/360043700632-Discords-engagemang-for-en-saker-och-trygg-upplevelse, Accessed: December 2020

[5] Discord, *Bug Bounty*, [Online]. Available: https://discord.com/security, Accessed: December 2020

[6] J. Sanati, *Top 10 reasons to setup a client-server network*, Maj 2011, [Online]. Available: https://intel.ly/37UpFD9, Accessed: December 2020

[7] Andrew Chi-Chih Yao. 1979. Some complexity questions related to distributive computing(Preliminary Report). In Proceedings of the eleventh annual ACM symposium on Theory of computing (STOC '79). Association for Computing Machinery, New York, NY, USA, 209–213. DOI:https://doi.org/10.1145/800135.804414

[8] Ashok K. Chandra, Merrick L. Furst, and Richard J. Lipton. 1983. Multi-party protocols. In Proceedings of the fifteenth annual ACM symposium on Theory of computing (STOC '83). Association for Computing Machinery, New York, NY, USA, 94–99. DOI:https://doi.org/10.1145/800061.808737

[9] Microsoft, *TCPView v3.05*, Juli 2011, [Online]. Available: https://docs.microsoft.com/en-us/sysinternals/downloads/tcpview, Accessed: December 2020

[10] *https://bimalchand.com.np/difference-between-client-server-and-peer-to-peer-network/*, Bildkälla: Difference between Client-Server and Peer-to-Peer Network, Juli 2019

[11] NetworkLessons, *TCP Window Size Scaling*, [Online]. Available: https://networklessons.com/cisco/ccie-routing-switching-written/tcp-window-size-scaling, Accessed: December 2020

[12] [RFC2018] Mathis, M., Mahdavi, J., Floyd, S. and A. Romanow, *TCP Selective Acknowledgement Options*, Oktober 1996, [Online]. Available: https://tools.ietf.org/html/rfc2018, Accessed: December 2020.

[13] J. Vass, *How Discord Handles Two and Half Million Concurrent Voice Users using WebRTC*, Discord, September 2018, [Online]. Available: https://blog.discord.com/how-discord-handles-two-and-half-million-concurrent-voice-users-using-webrtc-ce01c3187429, Accessed: December 2020