UNIVERZITET U NIŠU

ELEKTRONSKI FAKULTET

# Network Traffic Analyzer

Seminarski rad

Studijski program: Računarstvo i informatika

Modul: Softversko inženjerstvo

Mentor:

Prof. dr Bratislav Predić

Student:

Vuk Cvetković 1667

Niš, Jul 2024.

# Sadržaj

# Uvod

Analiza mrežnog saobraćaja predstavlja ključnu komponentu za održavanje i optimizaciju performansi, sigurnosti i pouzdanosti savremenih mreža. Mrežni saobraćaj obuhvata sve podatke koji se razmenjuju između različitih uređaja povezanih u mrežu, bilo da se radi o malim lokalnim mrežama (LAN) ili velikim globalnim mrežama poput Interneta. U današnjem svetu, gde su mreže temelj komunikacija, poslovanja i svakodnevnog života, razumevanje i analiza mrežnog saobraćaja postaju sve važniji.

### Značaj analize mrežnog saobraćaja

Analiza mrežnog saobraćaja je proces prikupljanja, pregleda i interpretacije mrežnih podataka kako bi se stekao uvid u rad mreže. Ovaj proces omogućava identifikaciju potencijalnih problema, praćenje performansi, optimizaciju mrežnih resursa i detekciju sigurnosnih pretnji. Na primer, analiza mrežnog saobraćaja može pomoći u identifikaciji zagušenja mreže, neautorizovanog pristupa ili napada, te u optimizaciji protoka podataka kroz mrežu.

### Tehnologije i alati za analizu

Postoji mnogo alata i tehnologija koje se koriste za analizu mrežnog saobraćaja, a među najpoznatijima su Wireshark, tcpdump, Pyshark i Scapy. Ovi alati omogućavaju prikupljanje i analizu paketa podataka koji prolaze kroz mrežu, pružajući detaljan uvid u različite protokole, komunikacije i anomalije.

### PCAP format i njegova važnost

Jedan od najvažnijih formata za prikupljanje mrežnih podataka je PCAP (Packet Capture) format. PCAP datoteke sadrže snimke mrežnog saobraćaja, omogućavajući analizu svake komunikacije u mreži do najsitnijih detalja. Ovaj format omogućava standardizovano prikupljanje i razmenu mrežnih podataka, što olakšava njihovu analizu i interpretaciju. PCAP datoteke se često koriste u alatima kao što su Wireshark i tcpdump, omogućavajući korisnicima da pregledaju, filtriraju i analiziraju mrežne pakete.

### Ciljevi ovog dokumenta

Ovaj dokument ima za cilj da pruži pregled analize mrežnog saobraćaja, sa posebnim fokusom na PCAP format i njegovu upotrebu. Dokument će biti podeljen na dva dela:

Teorijski deo, koji će obuhvatiti osnove mrežnog saobraćaja, metode i alate za analizu, kao i detalje o PCAP formatu.

Praktični deo, koji će opisati implementaciju aplikacije za analizu mrežnog saobraćaja korišćenjem Python-a, Tkinter-a i Pyshark-a.

# Mrežni saobraćaj

Mrežni saobraćaj predstavlja ključni aspekt modernih komunikacionih mreža. Razumevanje mrežnog saobraćaja je od suštinskog značaja za efikasno upravljanje mrežama, optimizaciju performansi i osiguranje bezbednosti podataka.

## Definicija mrežnog saobraćaja

Mrežni saobraćaj se odnosi na količinu podataka koji se kreću kroz računarske mreže u određenom vremenskom periodu. Podaci se prenose u obliku paketa koji putuju između izvora i odredišta preko različitih mrežnih medija, uključujući kablove, bežične veze, optička vlakna i druge tehnologije. Mrežni saobraćaj može obuhvatiti sve vrste komunikacija, od jednostavnih e-mail poruka do složenih video prenosa i IoT podataka.

## Tipovi mrežnog saobraćaja

Mrežni saobraćaj se može klasifikovati prema različitim kriterijumima, uključujući način na koji se paketi prenose kroz mrežu. Glavne vrste mrežnog saobraćaja su:

### Unicast saobraćaj

Unicast saobraćaj podrazumeva direktnu komunikaciju između dva mrežna uređaja, gde jedan uređaj šalje podatke drugom specifičnom uređaju. Ova vrsta saobraćaja je najčešća u mrežama i koristi se za većinu svakodnevnih aktivnosti kao što su surfovanje internetom, e-mail komunikacija i pristupanje web serverima.

Primer: Kada korisnik pretražuje web stranicu, njegov računar šalje unicast zahtev ka serveru koji hostuje tu stranicu, a server odgovara slanjem podataka nazad korisnikovom računaru.

### Multicast saobraćaj

Multicast saobraćaj uključuje slanje podataka sa jednog izvora ka grupi odredišta koja su članovi multicast grupe. Ova vrsta saobraćaja se koristi za aplikacije gde više korisnika treba da primi iste podatke istovremeno, kao što su video konferencije, strimovanje medija i slanje ažuriranja softvera.

Primer: Strimovanje uživo video prenosa gde se jedan video stream šalje svim korisnicima koji su se prijavili za gledanje tog prenosa.

### Broadcast saobraćaj

Broadcast saobraćaj podrazumeva slanje podataka sa jednog izvora ka svim uređajima u mreži. Ova vrsta saobraćaja se koristi za slanje informacija koje su relevantne za sve uređaje u mreži, kao što su ARP ili DHCP request.

Primer: Kada uređaj prvi put pristupa mreži, može koristiti broadcast saobraćaj da bi otkrio mrežni gateway pomoću ARP protokola.

## Protokoli u mrežnom saobraćaju

Mrežni protokoli su skup pravila koja definišu kako se podaci prenose kroz mrežu. Postoje mnogi protokoli koji se koriste u mrežnom saobraćaju, a najznačajniji među njima su:

| | |
|---|---|
| **TCP (Transmission Control Protocol)** | TCP je pouzdan, konekcijski orijentisan protokol koji se koristi za prenos podataka gde je važna ispravnost i redosled podataka. TCP osigurava da svi podaci stignu na odredište bez grešaka i u ispravnom redosledu, koristeći mehanizme za potvrdu prijema i ponovno slanje izgubljenih paketa.<br>**Primer**: Prilikom preuzimanja datoteke sa interneta, TCP osigurava da svi delovi datoteke stignu tačno i bez grešaka. |
| **UDP (User Datagram Protocol)** | UDP je nepouzdan, bezkonekcijski protokol koji se koristi za prenose gde je važna brzina, a ne nužno tačnost ili redosled podataka. UDP omogućava slanje podataka bez potvrde prijema, što ga čini bržim, ali manje pouzdanim od TCP-a.<br>**Primer**: Prilikom strimovanja video sadržaja uživo, gde je važna brzina prenosa i kašnjenje, a gubitak nekoliko paketa neće značajno uticati na korisničko iskustvo. |
| **ICMP (Internet Control Message Protocol)** | ICMP se koristi za slanje dijagnostičkih i kontrolnih poruka između uređaja u mreži. Ovaj protokol se često koristi za alate kao što su ping i traceroute, koji omogućavaju testiranje mrežnih konekcija i putanja.<br>**Primer**: Kada korisnik želi da proveri da li je određeni server dostupan, može koristiti ping komandu koja šalje ICMP Echo Request poruku serveru i čeka na ICMP Echo Reply. |

### Ostali protokoli

Pored glavnih protokola, postoji mnogo drugih protokola koji igraju ključne uloge u mrežnom saobraćaju:

| | |
|---|---|
| **HTTP/HTTPS (HyperText Transfer Protocol/Secure)** | Protokol za prenos hiperteksta se koristi za prenos web stranica i drugih resursa preko interneta. HTTPS je sigurnija verzija HTTP-a koja koristi SSL/TLS enkripciju za zaštitu podataka tokom prenosa. |
| **FTP (File Transfer Protocol)** | FTP se koristi za prenos datoteka između računara u mreži. Ovaj protokol omogućava korisnicima da prenose datoteke sa servera na svoj računar i obrnuto. |
| **SMTP (Simple Mail Transfer Protocol)** | SMTP se koristi za slanje e-mail poruka između servera. Ovaj protokol osigurava da e-mail poruke stignu do svojih odredišta putem interneta. |
| **DNS (Domain Name System)** | DNS se koristi za prevođenje domena imena u IP adrese. Kada korisnik unese web adresu u pregledač, DNS server prevodi tu |

adresu u odgovarajuću IP adresu koja se koristi za uspostavljanje konekcije.

Ovi protokoli, zajedno sa mnogim drugima, čine temelj komunikacije u modernim mrežama, omogućavajući prenos podataka, komunikaciju i pružanje različitih mrežnih usluga. Razumevanje ovih protokola je ključno za analizu mrežnog saobraćaja, dijagnostiku problema i optimizaciju mrežnih performansi.

# Analiza mrežnog saobraćaja

Analiza mrežnog saobraćaja je kritična za održavanje i unapređenje mrežnih sistema. Ona omogućava dubinsko razumevanje načina na koji podaci prolaze kroz mrežu, identifikaciju potencijalnih problema, poboljšanje performansi, i povećanje sigurnosti. U nastavku su dati primeri gde je važna analiza mrežnog saobraćaja:

| | |
|---|---|
| **Dijagnostika mrežnih problema** | Analiza mrežnog saobraćaja pomaže administratorima mreže da brzo identifikuju i reše probleme poput uskih grla, kašnjenja, gubitka paketa i prekida u komunikaciji. Detaljna analiza paketa može otkriti uzroke problema, bilo da su u pitanju loše konfiguracije, neadekvatna oprema ili preopterećenje mreže.<br><br>Primer: Ako korisnici prijavljuju spor internet, analiza mrežnog saobraćaja može pokazati tačno gde se u mreži javlja usko grlo i omogućiti brzo rešavanje problema. |
| **Sigurnost** | Analiza mrežnog saobraćaja je ključna za otkrivanje i sprečavanje sigurnosnih pretnji. Praćenjem i analizom saobraćaja, administratori mogu detektovati neovlašćene aktivnosti, napade, curenje podataka i druge sigurnosne incidente. Analiza može otkriti neobične obrasce saobraćaja koji mogu ukazivati na prisustvo zlonamernog softvera ili pokušaj proboja.<br><br>Primer: Identifikacija neovlašćenog pristupa putem analize neobičnih pokušaja logovanja ili nepoznatih IP adresa koje pokušavaju da pristupe mreži. |
| **Optimizacija performansi** | Analiza mrežnog saobraćaja omogućava optimizaciju performansi mreže tako što identifikuje nepotrebne ili neefikasne protoke podataka. Pomoću analize, moguće je bolje iskoristiti mrežne resurse, optimizovati postavke rutiranja i balansirati opterećenje kako bi se poboljšala ukupna efikasnost mreže.<br><br>Primer: Identifikacija i eliminacija nepotrebnog multicast ili broadcast saobraćaja koji opterećuje mrežu. |

## Metode i alati za analizu mrežnog saobraćaja

Postoji nekoliko metoda i alata koji se koriste za analizu mrežnog saobraćaja. Najpopularniji među njima su Wireshark, tcpdump i Pyshark. Svaki od ovih alata ima svoje prednosti i specifične primene.

### Wireshark

Wireshark je jedan od najpopularnijih alata za analizu mrežnog saobraćaja. To je besplatan, open-source alat koji omogućava korisnicima da pregledaju i analiziraju pakete podataka na vrlo detaljan način. Wireshark podržava veliki broj protokola i pruža korisnički interfejs koji omogućava lako pretraživanje i filtriranje podataka.

Karakteristike:

- Pregled i analiza paketa u realnom vremenu.
- Dekodiranje velikog broja mrežnih protokola.
- Snažni filteri za pretraživanje specifičnih podataka unutar saobraćaja.
- Vizuelizacija saobraćaja kroz grafove i statistike.

**tcpdump**

tcpdump je moćan alat za presretanje i analizu TCP/IP paketa. To je komandno-linijski alat koji omogućava snimanje i analizu mrežnog saobraćaja. tcpdump je idealan za brzu analizu i dijagnostiku mrežnih problema, posebno u Unix i Linux okruženjima.

Karakteristike:

- Snimanje i prikaz paketa u realnom vremenu.
- Podrška za prilagođene filtere za precizno snimanje specifičnih tipova saobraćaja.
- Mogućnost snimanja saobraćaja u datoteke za kasniju analizu.
- Komandno-linijski interfejs pogodan za integraciju sa skriptama i automatizacijom.

**Pyshark**

Pyshark je Python biblioteka koja omogućava analizu mrežnog saobraćaja koristeći Wireshark-ov tshark alat u pozadini. Pyshark pruža jednostavan način za automatsku analizu i manipulaciju mrežnim paketima unutar Python okruženja, što je korisno za razvoj prilagođenih alata za analizu mrežnog saobraćaja.

Karakteristike:

- Integracija sa Python-om za fleksibilnu analizu mrežnih podataka.
- Pristup svim funkcijama tshark-a unutar Python skripti.
- Mogućnost obrade i analize velikih količina podataka kroz skripte.
- Podrška za složene filtere i dekodiranje protokola.

# PCAP podaci

PCAP (Packet Capture) podaci predstavljaju snimljeni mrežni saobraćaj koji uključuje detaljne informacije o mrežnim paketima koji su prošli kroz mrežni interfejs. Ovi podaci se koriste za analizu mrežnog saobraćaja, dijagnostiku problema, sigurnosne analize i istraživanje mrežnih napada. PCAP podaci beleže sve aspekte mrežnih paketa, uključujući zaglavlja i korisničke podatke, omogućavajući detaljan uvid u mrežnu komunikaciju.

## Istorija i razvoj PCAP formata

PCAP format je razvijen kao deo projekata za mrežnu analizu, kao što su tcpdump i libpcap. PCAP format je ključna komponenta u svetu mrežne analize, omogućavajući snimanje i analizu mrežnog saobraćaja.

### Počeci i razvoj tcpdump-a

Sve je počelo krajem 1980-ih, kada su istraživači sa Berkli univerziteta radili na razvoju alata za mrežnu dijagnostiku. Godine 1988. predstavljen je tcpdump, jedan od prvih alata za presretanje paketa. Tcpdump je razvijen kao deo BSD (Berkeley Software Distribution) UNIX operativnog sistema. Njegova glavna funkcija bila je presretanje i prikaz mrežnih paketa koji prolaze kroz mrežni interfejs računara. Ovo je omogućilo mrežnim administratorima i istraživačima da analiziraju mrežni saobraćaj i identifikuju probleme u mreži.

### Razvoj libpcap-a

Kako se potreba za naprednijom mrežnom analizom povećavala, razvijena je biblioteka libpcap. Libpcap je nastao kao odgovor na potrebu za standardizovanim načinom snimanja i filtriranja mrežnih paketa na Unix operativnim sistemima. Biblioteka je pružila osnovu za tcpdump, omogućavajući mu da funkcioniše efikasnije i sa više fleksibilnosti. Libpcap je postao ključna komponenta u mnogim alatima za mrežnu analizu jer je omogućio jednostavan pristup i manipulaciju mrežnim paketima.

### PCAP kao standard

S vremenom, PCAP format se razvio u kao standard za snimanje mrežnog saobraćaja. Njegova jednostavnost i interoperabilnost učinili su ga popularnim među razvojnim timovima koji su želeli da izgrade alate za mrežnu analizu. PCAP fajlovi sadrže sirove podatke mrežnih paketa, uključujući metapodatke kao što su vremenski pečati, IP adrese, portovi i protokoli.

### Evolucija i primena

Tokom godina, PCAP format je evoluirao kako bi zadovoljio rastuće potrebe mrežne analize. Sa razvojem sve složenijih mrežnih infrastruktura, PCAP format je proširen kako bi uključio dodatne informacije i metapodatke. Ova fleksibilnost omogućila je njegovu široku primenu u različitim scenarijima, uključujući sigurnosnu analizu, dijagnostiku mrežnih problema, optimizaciju mrežnih performansi i istraživanje mrežnih protokola.

PCAP format je usvojen od strane mnogih komercijalnih i open-source alata. Wireshark, jedan od najpoznatijih alata za analizu mrežnog saobraćaja, koristi PCAP format za snimanje i analizu

paketa. Pored toga, brojni drugi alati kao što su Snort, Zeek (ranije Bro), i različiti IDS/IPS sistemi oslanjaju se na PCAP za prikupljanje i analizu mrežnih podataka.

**Proširenja i kompatibilnost**

Kako je mrežna tehnologija napredovala, pojavila su se i proširenja PCAP formata. Jedno od značajnih proširenja je PCAPng (PCAP Next Generation) format, koji pruža bolju podršku za moderne mreže i dodatne funkcionalnosti, uključujući podršku za više interfejsa, dodatne metapodatke, i napredne opcije filtriranja. PCAPng format je unazad kompatibilan sa originalnim PCAP formatom, što omogućava lakšu tranziciju i interoperabilnost između različitih alata.

Istorija i razvoj PCAP formata odražavaju njegovu vitalnu ulogu u mrežnoj analizi. Od svojih skromnih početaka sa tcpdump-om i libpcap-om na Berkliju, PCAP je evoluirao u globalni standard koji omogućava detaljnu analizu mrežnog saobraćaja. Uz sve ove prednosti, PCAP format će verovatno nastaviti da se razvija i prilagođava budućim izazovima mrežnog okruženja, ostajući ključna komponenta u svetu mrežne analize.

## Prednosti i nedostaci korišćenja PCAP podataka

**Prednosti:**

- Detaljna analiza: PCAP podaci omogućavaju detaljan uvid u mrežnu komunikaciju, uključujući sve slojeve mrežnog modela.
- Dijagnostika problema: Snimanje i analiza PCAP podataka pomažu u brzom identifikovanju i rešavanju mrežnih problema.
- Sigurnosna analiza: PCAP podaci omogućavaju identifikaciju i analizu sigurnosnih pretnji, kao što su neovlašćeni pristupi i napadi.
- Kompatibilnost: PCAP format je široko prihvaćen i podržan od strane mnogih alata za mrežnu analizu.

**Nedostaci:**

- Veliki fajlovi: Snimanje dugotrajnog ili obimnog mrežnog saobraćaja može rezultirati velikim PCAP datotekama koje zauzimaju mnogo prostora na disku.
- Privatnost: Snimanje mrežnog saobraćaja može uključivati privatne i osetljive podatke, što zahteva pažljivo upravljanje i zaštitu.
- Performanse: Snimanje i analiza velikog broja paketa može biti zahtevno za računar i može uticati na performanse mreže i računara.

## Formati i struktura PCAP datoteka

PCAP datoteke sastoje se od zaglavlja i niza zapisa o paketima. Svaka PCAP datoteka počinje globalnim zaglavljem koje opisuje format datoteke, uključujući verziju formata, timestamp, maksimalnu veličinu paketa i tip mreže. Nakon globalnog zaglavlja sledi niz zapisa o paketima, pri čemu svaki zapis sadrži metapodatke o paketu (kao što su timestamp, veličina paketa) i sam sadržaj paketa.

**Struktura PCAP datoteke:**

Globalno zaglavlje: Opisuje format i parametre snimanja.

- magic_number: Identifikator formata datoteke.
- version_major: Glavna verzija formata.
- version_minor: Podverzija formata.
- thiszone: Korekcija vremenske zone.
- sigfigs: Tačnost timestamp-ova.
- snaplen: Maksimalna veličina paketa.
- network: Tip mreže.

Zapis o paketu: Sadrži metapodatke i sadržaj paketa.

- ts_sec: Timestamp (sekunde).
- ts_usec: Timestamp (mikrosekunde).
- incl_len: Stvarna veličina paketa.
- orig_len: Originalna veličina paketa.
- packet_data: Sadržaj paketa.

PCAP datoteke omogućavaju snimanje i analizu mrežnog saobraćaja na detaljnom nivou, pružajući mrežnim stručnjacima alat za dijagnostiku, sigurnosnu analizu i optimizaciju mrežnih performansi. Njihova široka prihvaćenost i podrška u različitim alatima čine ih osnovnim resursom u mrežnoj analizi.

# Prikupljanje i analiza PCAP podataka

Prikupljanje PCAP podataka je ključni korak u analizi mrežnog saobraćaja, jer omogućava beleženje i skladištenje mrežnih paketa za kasniju analizu.

## Alati za prikupljanje PCAP podataka

Dva najpopularnija alata za prikupljanje PCAP podataka su Wireshark i Pyshark.

Wireshark je najpoznatiji i najrasprostranjeniji alat za analizu mrežnog saobraćaja sa grafičkim korisničkim interfejsom. Omogućava korisnicima da presretnu i analiziraju mrežne pakete u realnom vremenu. Wireshark podržava veliki broj mrežnih protokola i omogućava detaljno pregledanje svake mrežne komunikacije.

Pyshark je Python biblioteka koja omogućava analizu mrežnog saobraćaja koristeći funkcionalnosti Wiresharka. Pyshark je posebno koristan za automatizaciju analize mrežnog saobraćaja i integraciju sa drugim Python skriptama i aplikacijama. Korišćenjem Pysharka, analitičari mogu programatski pristupati i analizirati mrežne pakete.

## Proces prikupljanja PCAP podataka

Prikupljanje PCAP podataka uključuje nekoliko koraka:

- Odabir mrežnog interfejsa: Prvi korak je identifikacija mrežnog interfejsa koji će biti korišćen za snimanje mrežnog saobraćaja. Mrežni interfejsi mogu biti fizički (kao što su Ethernet ili Wi-Fi adapteri) ili virtuelni (kao što su tunelski interfejsi).
- Definisanje filtera: Da bi se snimanje ograničilo samo na relevantan saobraćaj, korisnici mogu definisati filtere. Filteri mogu biti bazirani na različitim kriterijumima kao što su IP adrese, portovi, protokoli itd.
- Pokretanje snimanja: Nakon postavljanja mrežnog interfejsa i filtera, alat za prikupljanje PCAP podataka se pokreće da bi započeo snimanje mrežnog saobraćaja. Snimanje može trajati onoliko dugo koliko je potrebno da se prikupi dovoljno podataka za analizu.
- Snimanje podataka: Tokom snimanja, alat beleži sve mrežne pakete koji prolaze kroz definisani mrežni interfejs i čuva ih u PCAP datoteci. Ove datoteke se mogu kasnije koristiti za detaljnu analizu mrežnog saobraćaja.
- Zaustavljanje snimanja: Kada je prikupljena dovoljna količina podataka, snimanje se zaustavlja. PCAP datoteka sa snimljenim podacima može se otvoriti u alatu za analizu mrežnog saobraćaja kao što je Wireshark ili se može koristiti u Pyshark skriptama za dalju analizu.

## Alati za analizu PCAP podataka

Nakon što su PCAP podaci prikupljeni, sledeći korak je njihova analiza. Analiza PCAP podataka je ključna za razumevanje mrežnog saobraćaja, identifikaciju problema i otkrivanje potencijalnih sigurnosnih pretnji. Neki od najvažnijih alata za analizu PCAP podataka uključuju Wireshark, Pyshark i Scapy.

Wireshark je sveobuhvatan alat za analizu mrežnog saobraćaja koji omogućava korisnicima da pregledaju i analiziraju snimljene PCAP datoteke. Koristeći Wireshark, analitičari mogu detaljno pregledati svaki mrežni paket, dekodirati protokole, pratiti tokove komunikacije i identifikovati potencijalne probleme.

Pyshark je moćan alat za analizu PCAP podataka kroz Python skripte. Pyshark omogućava automatizaciju procesa analize, što je posebno korisno za velike skupove podataka ili za redovne analize. Pyshark može ekstraktovati relevantne informacije iz PCAP datoteka i integrisati ih sa drugim Python alatima i bibliotekama.

Scapy je još jedan alat za analizu i generisanje mrežnog saobraćaja. Scapy je veoma fleksibilan i omogućava korisnicima da kreiraju sopstvene mrežne pakete, manipulišu njima i analiziraju mrežni saobraćaj. Scapy je posebno koristan za istraživanje i testiranje mrežnih protokola.

Korišćenjem ovih alata, analitičari mogu detaljno pregledati mrežni saobraćaj, identifikovati abnormalnosti, otkriti probleme i optimizovati performanse mreže. Analiza PCAP podataka je ključna za održavanje sigurnosti i efikasnosti mrežnih infrastruktura.

# Primena analize mrežnog saobraćaja

Analiza mrežnog saobraćaja pruža uvid u različite aspekte mrežnih komunikacija i omogućava prepoznavanje potencijalnih pretnji i nepravilnosti.

## Detekcija i analiza mrežnih napada

Analiza mrežnog saobraćaja igra ključnu ulogu u detekciji i analizi mrežnih napada. Kroz detaljnu analizu PCAP podataka, moguće je identifikovati različite vrste napada, uključujući:

| | |
|---|---|
| **DDoS napadi** | DDoS napadi su među najčešćim oblicima sajber napada. Analizom mrežnog saobraćaja može se otkriti neuobičajeno veliki broj zahteva prema određenom serveru, što je karakteristično za DDoS napade. Korišćenjem alata kao što su Wireshark ili Pyshark, analitičari mogu identifikovati izvore napada i tipove saobraćaja koji se koriste za preopterećenje sistema. |
| **Man-in-the-Middle napadi** | Ovi napadi uključuju presretanje i potencijalnu modifikaciju komunikacije između dva čvora u mreži. Analiza PCAP podataka omogućava prepoznavanje neovlašćenih modifikacija u komunikaciji ili prisustvo sumnjivih IP adresa koje se pojavljuju između validnih čvorova. Otkriće ovakvih anomalija pomaže u detekciji i prevenciji MITM napada. |

## Praćenje performansi mreže

Praćenje performansi mreže je esencijalno za osiguranje optimalnog funkcionisanja mrežne infrastrukture. Analizom mrežnog saobraćaja moguće je:

| | |
|---|---|
| **Identifikacija zagušenja** | Analiza PCAP podataka može ukazati na tačke u mreži gde dolazi do zagušenja. Ovo je korisno za optimizaciju mrežnih resursa i unapređenje protoka podataka. |
| **Merenje kašnjenja** | Analizom timestamp-ova u mrežnim paketima moguće je merenje latencije i identifikacija segmenata mreže gde dolazi do značajnih kašnjenja. Ovo pomaže u poboljšanju performansi mreže i kvaliteta usluge. |
| **Praćenje protoka** | Detaljna analiza protoka podataka kroz mrežu omogućava identifikaciju najčešće korišćenih servisa i aplikacija, što može pomoći u boljoj alokaciji resursa i kapaciteta mreže. |

## Dijagnostika mrežnih problema

Analiza mrežnog saobraćaja je ključna za efikasnu dijagnostiku mrežnih problema. Neki od uobičajenih scenarija gde analiza PCAP podataka može pomoći uključuju:

| | |
|---|---|
| **Identifikacija grešaka** | PCAP podaci omogućavaju detaljno praćenje svih mrežnih komunikacija, što pomaže u identifikaciji grešaka kao što su gubitak paketa, preusmeravanje i fragmentacija. |

| | |
|---|---|
| **Razumevanje komunikacionih problema** | Analizom komunikacionih tokova između različitih uređaja u mreži, moguće je identifikovati probleme sa povezivanjem, neuspešne pokušaje uspostavljanja veze ili neispravnu konfiguraciju mrežnih uređaja. |
| **Sigurnosna dijagnostika** | PCAP podaci mogu otkriti pokušaje neovlašćenog pristupa, prisustvo malicioznih aktivnosti ili pokušaje eksploatacije sigurnosnih propusta. Analitičari mogu koristiti ove podatke za brzo reagovanje i primenu odgovarajućih sigurnosnih mera. |

Analiza mrežnog saobraćaja je ključna komponenta mrežnog inženjeringa i administracije. Bez nje, održavanje visokih standarda performansi, sigurnosti i pouzdanosti mrežnih sistema bi bilo znatno teže. Kroz kontinuiranu analizu i praćenje mrežnog saobraćaja, organizacije mogu proaktivno detektovati i rešavati probleme, osiguravajući stabilno i sigurno mrežno okruženje.

# Aplikacija

Analiza mrežnog saobraćaja predstavlja ključni aspekt u održavanju i optimizaciji mrežnih sistema. Razvijena aplikacija za analizu mrežnog saobraćaja omogućava korisnicima da jednostavno učitaju i analiziraju mrežni saobraćaj snimljen u pcap-ng fajlovima. Kroz intuitivan grafički interfejs, aplikacija nudi mogućnost filtriranja i vizualizacije podataka, što olakšava detekciju problema i optimizaciju mrežnih performansi.

## Pregled funkcionalnosti aplikacije

Ključne funkcionalnosti aplikacije uključuju:

- Učitavanje pcap fajlova: Korisnici mogu jednostavno učitati fajlove sa mrežnim saobraćajem kako bi ih analizirali.
- Prikaz osnovnih informacija o paketima: Aplikacija prikazuje timestamp, IP adrese izvora i odredišta, dužinu paketa, i korišćene protokole.
- Filtriranje paketa: Omogućeno je filtriranje paketa po datumu, vremenu, IP adresama izvora i odredišta, kao i po protokolima.
- Vizualizacija podataka: Aplikacija prikazuje distribuciju protokola u obliku dijagrama (pie chart i bar chart).

### Kratak opis korišćenih tehnologija

**Python**

Python je programski jezik visokog nivoa, poznat po svojoj čitljivosti i jednostavnosti. Koristi se za širok spektar aplikacija, uključujući analizu podataka, veb razvoj, automatizaciju i mnoge druge oblasti. U ovoj aplikaciji, Python je korišćen zbog svoje bogate biblioteke i jednostavnosti integracije sa različitim alatima.

**Tkinter**

Tkinter je standardna Python biblioteka za izradu GUI (grafičkih korisničkih interfejsa). Omogućava kreiranje prozora, dugmadi, unosa teksta, tabela i drugih GUI elemenata na jednostavan način. U ovoj aplikaciji, Tkinter je korišćen za izradu korisničkog interfejsa koji omogućava interakciju sa korisnikom, uključujući učitavanje fajlova, prikaz rezultata analize i primenu filtera.

**Pyshark**

Pyshark je Python biblioteka koja omogućava parsiranje i analizu mrežnog saobraćaja iz pcap fajlova. Bazirana je na popularnom alatu Wireshark, koji je standard za analizu mrežnog saobraćaja. Pyshark omogućava jednostavan pristup informacijama o paketima, uključujući zaglavlja protokola, timestamp i mnogo više. U ovoj aplikaciji, Pyshark je korišćen za čitanje pcap fajlova i ekstrakciju relevantnih podataka za analizu i prikaz.

**Matplotlib**

Matplotlib je sveobuhvatna biblioteka za kreiranje statičkih, animiranih i interaktivnih vizualizacija u Pythonu. U ovoj aplikaciji, Matplotlib je korišćen za vizualizaciju distribucije

protokola u mrežnom saobraćaju. Aplikacija koristi Matplotlib za generisanje pie chart i bar chart grafikona, koji omogućavaju korisnicima da na intuitivan način vide kako su različiti protokoli zastupljeni u analiziranom saobraćaju.

## Funkcionalnosti aplikacije

U nastavku je dato objašnjenje rada aplikacije.

### Učitavanje fajla

Funkcija *open_file* otvara dijalog za izbor fajla pomoću Tkinterove funkcije filedialog.askopenfilename. Ova funkcija omogućava korisniku da izabere fajl sa ekstenzijama .pcapng ili .pcap, koji su tipični formati za PCAP fajlove. Kada korisnik izabere fajl i potvrdi izbor, putanja do tog fajla se čuva u promenljivoj file_path. Ako je fajl uspešno izabran (tj. file_path nije prazan), funkcija analyze_and_display se poziva sa putanjom do izabranog fajla kao argumentom, čime se pokreće analiza i prikaz sadržaja fajla.

```
def open_file():
    file_path = filedialog.askopenfilename(filetypes=[("PCAP files",
"*.pcapng *.pcap")])
    if file_path:
        analyze_and_display(file_path)
```

### Analiza i prikaz

```
def analyze_and_display(file_path):
    global current_file_path, all_packets
    current_file_path = file_path
    capture = read_pcap(file_path)
    all_packets = [pkt for pkt in capture]
    capture.close()
    result_tree.delete(*result_tree.get_children())

    reset_filters()
    results, protocol_counts = analyze_pcap(all_packets)
    for result in results:
        timestamp = result['Timestamp'].strftime("%Y-%m-%d %H:%M:%S")
        source_ip = result['Source IP']
        destination_ip = result['Destination IP']
        length = result['Length']
        protocols = result['Protocols']

        packet_id = result_tree.insert("", "end", text=timestamp, values=
                            (source_ip, destination_ip, length, protocols))

        for proto, data in result['Data'].items():
            proto_id = result_tree.insert(packet_id, "end", text=proto)
            for key, value in data.items():
                result_tree.insert(proto_id, "end", text=key, values=
                                                        (value,))

    visualize_data(protocol_counts, root)  # Prikaz grafikona
```

Funkcija *analyze_and_display* koristi se za analiziranje i prikazivanje sadržaja PCAP fajla. Na početku, postavlja dve globalne promenljive: *current_file_path*, koja čuva putanju do trenutnog

PCAP fajla, i *all_packets*, koja čuva listu svih paketa u fajlu. Zatim, postavlja *current_file_path* na vrednost *file_path*, čime se čuva putanja do izabranog fajla.

Funkcija zatim poziva *read_pcap* sa *file_path* kao argumentom, što omogućava čitanje sadržaja PCAP fajla i vraća objekat *capture* koji predstavlja učitane podatke. Svi paketi iz *capture* se učitavaju u listu *all_packets*.

Nakon učitavanja paketa, *capture* se zatvara kako bi se oslobodili resursi. Zatim se brišu svi prethodni rezultati iz *result_tree*, što je vizuelni prikaz rezultata u aplikaciji.

Poziva se *reset_filters* da bi se resetovali svi filteri pre analize novih podataka. Funkcija *analyze_pcap* se poziva sa listom *all_packets* i vraća analizirane rezultate i broj protokola.

Rezultati analize se dodaju u *result_tree*. Za svaki rezultat, izvlače se podaci kao što su timestamp, izvorna IP adresa, odredišna IP adresa, dužina paketa i protokoli. Ovi podaci se prikazuju u *result_tree* kao novi čvorovi.

Ako paket sadrži dodatne podatke po protokolima, dodaju se podčvorovi sa detaljima svakog protokola i odgovarajućim informacijama.

Na kraju, funkcija *visualize_data* se poziva sa brojem protokola i korenskim prozorom aplikacije (*root*) kako bi se prikazao grafički prikaz podataka.

### Čitanje sadržaja

```
def read_pcap(file_path):
    return pyshark.FileCapture(file_path)
```

Funkcija *read_pcap* koristi se za čitanje sadržaja PCAP fajla. Ona prima jedan argument, *file_path*, koji predstavlja putanju do PCAP fajla. Funkcija koristi Pyshark biblioteku za otvaranje i čitanje PCAP fajla. Pyshark-ova klasa FileCapture se koristi za kreiranje objekta koji omogućava pristup paketima u PCAP fajlu. Funkcija vraća taj objekat, omogućavajući dalju obradu paketa u drugim delovima programa.

### Analiza paketa iz PCAP fajla

```
def analyze_pcap(packets, start_date=None, end_date=None, source_ip="",
destination_ip="", protocolsString=""):
    results = []
    protocol_counts = {}
    global fromApplyFilter

    for i, packet in enumerate(packets, start=1):
        if (fromApplyFilter == True):
            if start_date and packet.sniff_time < start_date:
                continue
            if end_date and packet.sniff_time > end_date:
                continue
            if hasattr(packet, 'ip'):
                if source_ip != "" and packet.ip.src != source_ip:
                    continue
                if destination_ip != "" and packet.ip.dst !=
destination_ip:
                    continue
            if protocolsString != "":
                protocolsList = protocolsString.split(',')
```

```
                protocols = extract_all_protocols(packet).split(', ')
                found = False
                for protocol in protocolsList:
                    if protocol in protocols:
                        found = True
                        break
                if not found:
                    continue
            packet_data = {}
            if 'HTTP' in packet:
                packet_data['HTTP'] = extract_http_data(packet)
            if 'SSL' in packet:
                packet_data['HTTPS'] = extract_https_data(packet)
            if 'DNS' in packet:
                packet_data['DNS'] = extract_dns_data(packet)
            if 'FTP' in packet:
                packet_data['FTP'] = extract_ftp_data(packet)
            if 'SMTP' in packet:
                packet_data['SMTP'] = extract_smtp_data(packet)
            if 'ARP' in packet:
                packet_data['ARP'] = extract_arp_data(packet)
            if 'ICMP' in packet:
                packet_data['ICMP'] = extract_icmp_data(packet)
            if 'IP' in packet:
                packet_data['IP'] = extract_ip_data(packet)
            if 'ETH' in packet:
                packet_data['Ethernet'] = extract_ethernet_data(packet)
            if 'TCP' in packet:
                packet_data['TCP'] = extract_tcp_data(packet)
            if 'UDP' in packet:
                packet_data['UDP'] = extract_udp_data(packet)
            if 'FPP' in packet:
                packet_data['FPP'] = extract_fpp_data(packet)

            protocols = extract_all_protocols(packet).split(', ')
            for proto in protocols:
                if proto not in protocol_counts:
                    protocol_counts[proto] = 0
                protocol_counts[proto] += 1
            if packet_data:
                packet_info = {
                    'Packet Number': f"Packet {i}",
                    'Timestamp': packet.sniff_time,
                    'Length': packet.length,
                    'Source IP': packet.ip.src if hasattr(packet, 'ip')
                                                else 'N/A',
                    'Destination IP': packet.ip.dst if hasattr(packet, 'ip')
                                                    else 'N/A',
                    'Protocols': ', '.join(protocols),
                    'Data': packet_data
                }
                results.append(packet_info)
    return results, protocol_counts
```

Funkcija *analyze_pcap* analizira pakete iz PCAP fajla i omogućava filtriranje paketa prema zadatim kriterijumima. Prima sledeće argumente: *packets* (lista paketa), *start_date* (početni datum za filtriranje), *end_date* (krajnji datum za filtriranje), *source_ip* (izvorna IP adresa), *destination_ip* (odredišna IP adresa), i *protocolsString* (niz protokola za filtriranje).

Funkcija radi sledeće:

- Kreira prazne liste *results* za rezultate analize i *protocol_counts* za brojanje pojavljivanja protokola.
- Prolazi kroz svaki paket u listi *packets*.
- Ako je uključen filter (*fromApplyFilter == True*), proverava da li paket zadovoljava kriterijume filtera:
  - Timestamp paketa mora biti u okviru zadatih datuma.
  - Izvorna i odredišna IP adresa, ako su zadate, moraju odgovarati adresama paketa.
  - Paket mora sadržati bar jedan od zadatih protokola.
- Ekstraktuje podatke o protokolima iz paketa, ako su prisutni:
  - HTTP, HTTPS (SSL), DNS, FTP, SMTP, ARP, ICMP, IP, Ethernet, TCP, UDP, i FPP.
- Broji pojavljivanja svakog protokola u paketima.
- Ako paket sadrži relevantne podatke, kreira dictionary *packet_info* sa informacijama o paketu (broj paketa, vreme, dužina, izvorna i odredišna IP adresa, protokoli i ekstraktovani podaci).
- Dodaje *packet_info* u listu *results*.
- Na kraju, vraća listu results sa analiziranim podacima i dictionary *protocol_counts* sa brojem pojavljivanja svakog protokola.

### Ekstrakcija podataka

```
import base64


def extract_http_data(packet):
    data = {}
    if hasattr(packet.http, 'authorization'):
        if 'Basic' in packet.http.authorization:
            auth_info = packet.http.authorization.split('Basic ')[1]
            decoded_auth = base64.b64decode(auth_info).decode()
            data['HTTP Basic Auth'] = decoded_auth
    if hasattr(packet.http, 'host'):
        data['Host'] = packet.http.host
    if hasattr(packet.http, 'user_agent'):
        data['User Agent'] = packet.http.user_agent
    if hasattr(packet.http, 'request_uri'):
        data['Request URI'] = packet.http.request_uri
    if hasattr(packet.http, 'file_data'):
        data['Form Data'] = packet.http.file_data
    if hasattr(packet.http, 'cookie'):
        data['Cookie'] = packet.http.cookie
    if hasattr(packet.http, 'content_type'):
        data['Content Type'] = packet.http.content_type
    if hasattr(packet.http, 'response_code'):
        data['Response Code'] = packet.http.response_code
    if hasattr(packet.http, 'server'):
        data['Server'] = packet.http.server
    if hasattr(packet.http, 'referer'):
        data['Referer'] = packet.http.referer
    if hasattr(packet.http, 'accept_language'):
```

```
        data['Accept Language'] = packet.http.accept_language
    if hasattr(packet.http, 'accept_encoding'):
        data['Accept Encoding'] = packet.http.accept_encoding
    if hasattr(packet.http, 'content_length'):
        data['Content Length'] = packet.http.content_length
    return data

def extract_https_data(packet):
    data = {}
    if hasattr(packet.ssl, 'handshake_version'):
        data['SSL Version'] = packet.ssl.handshake_version
    if hasattr(packet.ssl, 'handshake_cipher_suite'):
        data['Cipher Suite'] = packet.ssl.handshake_cipher_suite
    if hasattr(packet.ssl, 'handshake_extensions_server_name'):
        data['Server Name'] = packet.ssl.handshake_extensions_server_name
    return data

def extract_dns_data(packet):
    data = {}
    if hasattr(packet.dns, 'qry_name'):
        data['DNS Query'] = packet.dns.qry_name
    if hasattr(packet.dns, 'a'):
        data['DNS Response'] = packet.dns.a
    if hasattr(packet.dns, 'qry_type'):
        data['Query Type'] = packet.dns.qry_type
    if hasattr(packet.dns, 'qry_class'):
        data['Query Class'] = packet.dns.qry_class
    return data

def extract_ftp_data(packet):
    data = {}
    if hasattr(packet.ftp, 'request'):
        data['FTP Request'] = packet.ftp.request
    if hasattr(packet.ftp, 'response'):
        data['FTP Response'] = packet.ftp.response
    if hasattr(packet.ftp, 'username'):
        data['Username'] = packet.ftp.username
    if hasattr(packet.ftp, 'password'):
        data['Password'] = packet.ftp.password
    return data

def extract_smtp_data(packet):
    data = {}
    if hasattr(packet.smtp, 'mail_from'):
        data['SMTP From'] = packet.smtp.mail_from
    if hasattr(packet.smtp, 'rcpt_to'):
        data['SMTP To'] = packet.smtp.rcpt_to
    if hasattr(packet.smtp, 'data'):
        data['SMTP Data'] = packet.smtp.data
    if hasattr(packet.smtp, 'subject'):
        data['Subject'] = packet.smtp.subject
    return data

def extract_arp_data(packet):
    data = {}
    if hasattr(packet.arp, 'src_proto_ipv4'):
        data['ARP Source IP'] = packet.arp.src_proto_ipv4
    if hasattr(packet.arp, 'dst_proto_ipv4'):
        data['ARP Destination IP'] = packet.arp.dst_proto_ipv4
    if hasattr(packet.arp, 'hw_src'):
        data['Hardware Source'] = packet.arp.hw_src
```

```
    if hasattr(packet.arp, 'hw_dst'):
        data['Hardware Destination'] = packet.arp.hw_dst
    if hasattr(packet.arp, 'opcode'):
        data['Opcode'] = packet.arp.opcode
    return data

def extract_icmp_data(packet):
    data = {}
    if hasattr(packet.icmp, 'type'):
        data['ICMP Type'] = packet.icmp.type
    if hasattr(packet.icmp, 'code'):
        data['ICMP Code'] = packet.icmp.code
    if hasattr(packet.icmp, 'seq'):
        data['Sequence'] = packet.icmp.seq
    if hasattr(packet.icmp, 'checksum'):
        data['Checksum'] = packet.icmp.checksum
    return data

def extract_ip_data(packet):
    data = {}
    if hasattr(packet.ip, 'src'):
        data['IP Source'] = packet.ip.src
    if hasattr(packet.ip, 'dst'):
        data['IP Destination'] = packet.ip.dst
    if hasattr(packet.ip, 'ttl'):
        data['TTL'] = packet.ip.ttl
    if hasattr(packet.ip, 'len'):
        data['IP Packet Length'] = packet.ip.len
    if hasattr(packet.ip, 'flags'):
        data['IP Flags'] = packet.ip.flags
    if hasattr(packet.ip, 'id'):
        data['IP ID'] = packet.ip.id
    if hasattr(packet.ip, 'tos'):
        data['Type of Service'] = packet.ip.tos
    if hasattr(packet.ip, 'proto'):
        data['Protocol'] = packet.ip.proto
    return data

def extract_ethernet_data(packet):
    data = {}
    if hasattr(packet.eth, 'src'):
        data['Ethernet Source'] = packet.eth.src
    if hasattr(packet.eth, 'dst'):
        data['Ethernet Destination'] = packet.eth.dst
    if hasattr(packet.eth, 'type'):
        data['Ethernet Type'] = packet.eth.type
    if hasattr(packet.eth, 'src_resolved'):
        data['Source Resolved'] = packet.eth.src_resolved
    if hasattr(packet.eth, 'dst_resolved'):
        data['Destination Resolved'] = packet.eth.dst_resolved
    return data

def extract_tcp_data(packet):
    data = {}
    if hasattr(packet.tcp, 'srcport'):
        data['TCP Source Port'] = packet.tcp.srcport
    if hasattr(packet.tcp, 'dstport'):
        data['TCP Destination Port'] = packet.tcp.dstport
    if hasattr(packet.tcp, 'seq'):
        data['TCP Sequence Number'] = packet.tcp.seq
    if hasattr(packet.tcp, 'ack'):
```

```
        data['TCP Acknowledgment Number'] = packet.tcp.ack
    if hasattr(packet.tcp, 'flags'):
        data['TCP Flags'] = packet.tcp.flags
    if hasattr(packet.tcp, 'window_size'):
        data['Window Size'] = packet.tcp.window_size
    if hasattr(packet.tcp, 'checksum'):
        data['Checksum'] = packet.tcp.checksum
    if hasattr(packet.tcp, 'urgent_pointer'):
        data['Urgent Pointer'] = packet.tcp.urgent_pointer
    if hasattr(packet.tcp, 'options'):
        data['Options'] = packet.tcp.options
    return data

def extract_udp_data(packet):
    data = {}
    if hasattr(packet.udp, 'srcport'):
        data['UDP Source Port'] = packet.udp.srcport
    if hasattr(packet.udp, 'dstport'):
        data['UDP Destination Port'] = packet.udp.dstport
    if hasattr(packet.udp, 'length'):
        data['Length'] = packet.udp.length
    if hasattr(packet.udp, 'checksum'):
        data['Checksum'] = packet.udp.checksum
    return data

def extract_fpp_data(packet):
    data = {}
    if hasattr(packet.fpp, 'session_id'):
        data['Session ID'] = packet.fpp.session_id
    if hasattr(packet.fpp, 'src_ip'):
        data['Source IP'] = packet.fpp.src_ip
    if hasattr(packet.fpp, 'dst_ip'):
        data['Destination IP'] = packet.fpp.dst_ip
    if hasattr(packet.fpp, 'src_port'):
        data['Source Port'] = packet.fpp.src_port
    if hasattr(packet.fpp, 'dst_port'):
        data['Destination Port'] = packet.fpp.dst_port
    if hasattr(packet.fpp, 'protocol'):
        data['Protocol'] = packet.fpp.protocol
    if hasattr(packet.fpp, 'length'):
        data['Length'] = packet.fpp.length
    if hasattr(packet.fpp, 'checksum'):
        data['Checksum'] = packet.fpp.checksum
    if hasattr(packet.fpp, 'timestamp'):
        data['Timestamp'] = packet.fpp.timestamp
    if hasattr(packet.fpp, 'flags'):
        data['Flags'] = packet.fpp.flags
    return data

def extract_tls_data(packet):
    data = {}
    if hasattr(packet.ssl, 'handshake_version'):
        data['SSL Version'] = packet.ssl.handshake_version
    if hasattr(packet.ssl, 'handshake_cipher_suite'):
        data['Cipher Suite'] = packet.ssl.handshake_cipher_suite
    if hasattr(packet.ssl, 'handshake_extensions_server_name'):
        data['Server Name'] = packet.ssl.handshake_extensions_server_name
    if hasattr(packet.ssl, 'handshake_certificates'):
        data['Certificates'] = packet.ssl.handshake_certificates
    return data
```

Ovaj blok Python koda sadrži nekoliko funkcija koje ekstraktuju specifične podatke iz različitih vrsta paketa (HTTP, HTTPS, DNS, FTP, SMTP, ARP, ICMP, IP, Ethernet, TCP, UDP, FPP) koji se mogu naći u PCAP snimku.

- extract_http_data(packet):
    - Ekstraktuje HTTP podatke iz paketa ako su dostupni, kao što su HTTP Basic Authentication informacije, Host, User Agent, Request URI, Form Data, Cookie, Content Type, Response Code, Server, Referer, Accept Language, Accept Encoding, i Content Length.
- extract_https_data(packet):
    - Ekstraktuje HTTPS podatke iz paketa ako su dostupni, kao što su verzija SSL-a, kriptografski algoritam i Server Name.
- extract_dns_data(packet):
    - Ekstraktuje DNS podatke iz paketa ako su dostupni, kao što su DNS upit, odgovor, tip upita i klasa upita.
- extract_ftp_data(packet):
    - Ekstraktuje FTP podatke iz paketa ako su dostupni, kao što su FTP zahtev, odgovor, korisničko ime i lozinka.
- extract_smtp_data(packet):
    - Ekstraktuje SMTP podatke iz paketa ako su dostupni, kao što su SMTP pošiljalac, primalac, podaci i naslov poruke.
- extract_arp_data(packet):
    - Ekstrahuje ARP podatke iz paketa ako su dostupni, kao što su ARP izvorna IP adresa, odredišna IP adresa, hardverska adresa izvora i odredišta, i opcija.
- extract_icmp_data(packet):
    - Ekstraktuje ICMP podatke iz paketa ako su dostupni, kao što su ICMP tip, kod, sekvenca i checksum.
- extract_ip_data(packet):
    - Ekstraktuje IP podatke iz paketa ako su dostupni, kao što su izvorna i odredišna IP adresa, TTL, dužina paketa, zastavice, ID, Tip usluge i Protokol.
- extract_ethernet_data(packet):
    - Ekstraktuje Ethernet podatke iz paketa ako su dostupni, kao što su izvorna i odredišna MAC adresa, tip Ethernet okvira i rezolucija izvorne i odredišne adrese.
- extract_tcp_data(packet):
    - Ekstraktuje TCP podatke iz paketa ako su dostupni, kao što su izvorni i odredišni port, sekvenca, potvrda, flags, veličina prozora, checksum.
- extract_udp_data(packet):
    - Ekstraktuje UDP podatke iz paketa ako su dostupni, kao što su izvorni i odredišni port, dužina i checksum.
- extract_fpp_data(packet):

  - Ekstraktuje FPP (Fast Path Packet) podatke iz paketa ako su dostupni, kao što su ID sesije, izvorna i odredišna IP adresa, izvorni i odredišni port, protokol, dužina, checksum, timestamp i zastavice.
- extract_tls_data(packet):
  - Ekstraktuje TLS podatke iz paketa ako su dostupni, kao što su verzija TLS-a, kriptografski algoritam, Server Name i sertifikati.

Svaka od ovih funkcija proverava postojanje odgovarajućih atributa u paketu (npr. packet.http, packet.ssl, packet.dns) i ako su atributi prisutni, dodaje ih u dictionary *data* koji se zatim vraća kao rezultat funkcije.

### Vizualizacija

```
def visualize_data(protocol_counts, root):
    global canvas
    if canvas:
        canvas.get_tk_widget().pack_forget()

    protocols = list(protocol_counts.keys())
    counts = list(protocol_counts.values())

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

    # Pie chart
    ax1.pie(counts, labels=protocols, autopct='%1.1f%%', startangle=140,
            colors=plt.cm.Paired(range(len(protocols))))
    ax1.axis('equal')
    ax1.set_title('Protocol Distribution - Pie Chart')

    # Bar chart
    ax2.bar(protocols, counts, color='skyblue')
    ax2.set_xlabel('Protocol')
    ax2.set_ylabel('Count')
    ax2.set_title('Protocol Distribution - Bar Chart')

    canvas = FigureCanvasTkAgg(fig, master=root)
    canvas.draw()
    canvas.get_tk_widget().pack()
```

Funkcija *visualize_data* je namenjena vizualizaciji distribucije protokola u PCAP snimku kroz dva osnovna tipa grafikona: pie chart i bar chart.

Prvo, funkcija proverava postoji li već postojeći canvas za crtanje grafikona. Ako postoji, sklanja ga sa prikaza kako bi se mogao ažurirati novim grafikonom. Zatim, iz *protocol_counts* rečnika izvlači listu protokola.

Nakon toga, funkcija kreira *subplot* sa dva grafikona koristeći matplotlib. Pie chart dijagram je konfigurisan da prikazuje procenat distribucije protokola. Ovaj dijagram daje vizuelni pregled učešća svakog protokola u fajlu.

Bar chart dijagram prikazuje broj pojavljivanja svakog protokola na osnovu njihovih naziva i brojeva.

Na kraju, funkcija koristi *FigureCanvasTkAgg* za crtanje grafikona u *Tkinter* aplikaciji, omogućavajući korisnicima da interaktivno istražuju i interpretiraju distribuciju protokola u njihovom PCAP snimku.

### Filtriranje

```
def apply_filter():
    if current_file_path:
        global fromApplyFilter
        fromApplyFilter = True

        start_date_value = start_date_entry.get_date()
        end_date_value = end_date_entry.get_date()

        start_time_value = start_time_entry.get()
        end_time_value = end_time_entry.get()
        if not start_time_value or start_time_value == "":
            start_time_value = "00:00:00"
        if not end_time_value or end_time_value == "":
            end_time_value = "23:59:59"
        start_datetime = datetime.combine(start_date_value,
datetime.strptime(start_time_value, "%H:%M:%S").time())
        end_datetime = datetime.combine(end_date_value,
datetime.strptime(end_time_value, "%H:%M:%S").time())

        results, protocol_counts = analyze_pcap(all_packets,
start_datetime, end_datetime, source_ip_entry.get(), dest_ip_entry.get(),
protocols_entry.get())
        fromApplyFilter = False
        result_tree.delete(*result_tree.get_children())

        for result in results:
            timestamp = result['Timestamp'].strftime("%Y-%m-%d %H:%M:%S")
            source_ip = result['Source IP']
            destination_ip = result['Destination IP']
            length = result['Length']
            protocols = result['Protocols']

            packet_id = result_tree.insert("", "end", text=timestamp,
values=(source_ip, destination_ip, length, protocols))

            for proto, data in result['Data'].items():
                proto_id = result_tree.insert(packet_id, "end", text=proto)
                for key, value in data.items():
                    result_tree.insert(proto_id, "end", text=key,
values=(value,))

        visualize_data(protocol_counts, root)  # Prikaz grafikona
```

Funkcija *apply_filter()* je zadužena za primenu filtera na postojeće podatke u aplikaciji za analizu mrežnog saobraćaja. Prvo, funkcija proverava da li postoji putanja do trenutnog PCAP fajla (*current_file_path*). Ako postoji, postavlja globalnu promenljivu *fromApplyFilter* na *True*, što označava da se primenjuju filteri.

Zatim, funkcija dobavlja vrednosti početnog i krajnjeg datuma i vremena iz odgovarajućih polja za unos (*start_date_entry*, *end_date_entry*, *start_time_entry*, *end_time_entry*). Ako vrednosti za vreme nisu unete ili su prazne, funkcija ih podrazumevano postavlja na "00:00:00" za

početno vreme i "23:59:59" za krajnje vreme. Nakon toga, kombinuje datum i vreme u objekte *datetime* za početak (*start_datetime*) i kraj (*end_datetime*) perioda koji će se koristiti kao filteri.

Poziva funkciju *analyze_pcap()* sa paketima (*all_packets*), prethodno određenim datumima i vremenima, kao i sa dodatnim filterima koji su uneti kao IP adrese izvora i odredišta (*source_ip_entry, dest_ip_entry*) i protokolima (*protocols_entry*). Dobijeni rezultati i brojanje protokola se čuvaju u promenljivama *results* i *protocol_counts*.

Nakon primene filtera, vraća se vrednost *fromApplyFilter* na *False*. Prethodni prikaz u stablu rezultata (*result_tree*) se briše, a zatim se rezultati prikazuju ponovo. Svaki rezultat se formatira i dodaje u *result_tree,* a zatim se vizualizuje distribucija protokola pozivom funkcije *visualize_data()*.

Ova funkcija omogućava korisnicima da filtriraju i analiziraju specifične segmente mrežnog saobraćaja na osnovu datuma, vremena, IP adresa i protokola, pružajući im detaljan uvid u relevantne informacije iz PCAP fajla.

## GUI

```
# Kreiranje GUI-ja
root = tk.Tk()
root.title("Network Traffic Analyzer")

frame = tk.Frame(root)
frame.pack(padx=10, pady=10, fill=tk.X)

open_button = tk.Button(frame, text="Open PCAP File", command=open_file,
bg='lightblue')
open_button.pack(pady=5)

filter_frame = tk.Frame(root)
filter_frame.pack(padx=10, pady=10, fill=tk.X)

start_date_label = tk.Label(filter_frame, text="Start Date")
start_date_label.grid(row=0, column=0)
start_date_entry = DateEntry(filter_frame, width=12, background='darkblue',
foreground='white', borderwidth=2)
start_date_entry.grid(row=1, column=0, padx=5, pady=5)

end_date_label = tk.Label(filter_frame, text="End Date")
end_date_label.grid(row=0, column=1)
end_date_entry = DateEntry(filter_frame, width=12, background='darkblue',
foreground='white', borderwidth=2)
end_date_entry.grid(row=1, column=1, padx=5, pady=5)

tk.Label(filter_frame, text="Start Time:").grid(row=0, column=2, padx=10,
pady=5)
start_time_entry = ttk.Entry(filter_frame)
start_time_entry.grid(row=0, column=3, padx=10, pady=5)

tk.Label(filter_frame, text="End Time:").grid(row=1, column=2, padx=10,
pady=5)
end_time_entry = ttk.Entry(filter_frame)
end_time_entry.grid(row=1, column=3, padx=10, pady=5)
```

```
source_ip_label = tk.Label(filter_frame, text="Source IP Filter")
source_ip_label.grid(row=0, column=4)
source_ip_entry = tk.Entry(filter_frame)
source_ip_entry.grid(row=1, column=4, padx=5, pady=5)

dest_ip_label = tk.Label(filter_frame, text="Destination IP Filter")
dest_ip_label.grid(row=0, column=5)
dest_ip_entry = tk.Entry(filter_frame)
dest_ip_entry.grid(row=1, column=5, padx=5, pady=5)

protocols_label = tk.Label(filter_frame, text="Protocols Filter")
protocols_label.grid(row=0, column=6)
protocols_entry = tk.Entry(filter_frame)
protocols_entry.grid(row=1, column=6, padx=5, pady=5)

apply_button = tk.Button(filter_frame, text="Apply Filter",
command=apply_filter, bg='lightgreen')
apply_button.grid(row=1, column=7, padx=5, pady=5)

result_frame = tk.Frame(root)
result_frame.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

result_tree = ttk.Treeview(result_frame)
result_tree.pack(fill=tk.BOTH, expand=True)

result_tree["columns"] = ("Source IP", "Destination IP", "Length",
"Protocols")
result_tree.heading("#0", text="Timestamp")
result_tree.heading("Source IP", text="Source IP")
result_tree.heading("Destination IP", text="Destination IP")
result_tree.heading("Length", text="Length")
result_tree.heading("Protocols", text="Protocols")

root.mainloop()
```

Ovaj kod kreira grafički korisnički interfejs (GUI). On se sastoji od sledećih delova:

- Glavni prozor (*root*):
  - Koristi se tk.Tk() za inicijalizaciju glavnog prozora aplikacije.
  - Postavlja se naslov prozora na "Network Traffic Analyzer".
- Frame za dugme koje otvara PCAP fajla (*open_button*):
  - Unutar glavnog prozora (*root*) se kreira Frame objekat (*frame*) koji služi kao kontejner za organizaciju elemenata.
  - Dugme "Open PCAP File" (*open_button*) se kreira unutar frame-a.
- Frame za filtere (*filter_frame*):
  - Drugi Frame objekat (*filter_frame*) se kreira unutar glavnog prozora (*root*).
  - *filter_frame* sadrži niz labela i polja za unos za filtere podataka: početni datum, krajnji datum, početno vreme, krajnje vreme, filter za izvorišnu IP adresu, filter za odredišnu IP adresu, i filter za protokole.
  - Svaki element je organizovan koristeći grid raspored unutar *filter_frame-a*.
- Dugme za primenu filtera (*apply_button*):
  - Kreira se dugme "Apply Filter" (*apply_button*) unutar *filter_frame-a*.
  - Dugme reaguje na klik korisnika pozivajući funkciju *apply_filter*.

- Frame za rezultate (*result_frame*):
    - Kreira se Frame objekat (*result_frame*) koji se pakuje unutar glavnog prozora (*root*).
    - *result_frame* sadrži *Treeview* objekat (*result_tree*) koji služi za prikaz rezultata analize mrežnog saobraćaja.
    - *result_tree* je konfigurisan da prikaže kolone za vreme, izvorišnu IP adresu, odredišnu IP adresu, dužinu i protokole.
- Pokretanje glavne petlje (*root.mainloop()*):
    - Pokreće se glavna petlja aplikacije koja čeka korisničke interakcije kao što su klikovi na dugmiće i unos podataka.

Ovaj kod zajedno formira korisnički interfejs za aplikaciju koja omogućava otvaranje PCAP fajlova, primenu filtera na podatke u fajlu i prikaz rezultata analize mrežnog saobraćaja u tabelarnom obliku.

# Zaključak

U ovom seminarskom radu objašnjeni su ključni aspekti mrežnog saobraćaja i metode njegove analize. Teorijski deo rada obuhvatio je detaljnu definiciju mrežnog saobraćaja, različite tipove protokola, kao i alate i metode za analizu mrežnog saobraćaja. Fokus je bio na PCAP formatu, njegovoj istoriji, načinu prikupljanja podataka i prednosti koje donosi u analizi mrežnog saobraćaja.

Praktični deo rada uključivao je razvoj aplikacije za analizu mrežnog saobraćaja korišćenjem PyShark biblioteke za ekstrakciju podataka iz PCAP datoteka i Tkinter biblioteke za razvoj grafičkog korisničkog interfejsa. Ova aplikacija omogućava korisnicima da učitaju PCAP datoteku, filtriraju podatke na osnovu različitih kriterijuma kao što su datum, vreme, izvorna i odredišna IP adresa, protokoli. Pored toga, aplikacija nudi mogućnost vizualizacije rezultata analize putem grafova.

Kroz razvoj ove aplikacije demonstriran je proces čitanja i analize PCAP podataka, uključujući ekstrakciju informacija o različitim mrežnim protokolima kao što su HTTP, HTTPS, DNS, FTP, SMTP, ARP, ICMP, IP, Ethernet, TCP i UDP. Vizualizacija podataka omogućava lakše razumevanje distribucije različitih protokola u saobraćaju, što može biti korisno za identifikaciju abnormalnosti i potencijalnih sigurnosnih pretnji.

Analiza mrežnog saobraćaja od suštinskog je značaja za sigurnost i efikasnost mrežnih sistema. Ovaj rad pruža osnovu za razumevanje kako se mrežni saobraćaj može analizirati i kako se dobijeni podaci mogu koristiti za unapređenje mrežne infrastrukture. Dalje istraživanje i razvoj alata za analizu mrežnog saobraćaja može doprineti boljem upravljanju mrežama i zaštiti od potencijalnih pretnji. Takođe, napredak u tehnologijama mašinskog učenja i veštačke inteligencije može dodatno unaprediti analizu mrežnog saobraćaja, omogućavajući automatizovano otkrivanje i reagovanje na anomalije i napade.

# Literatura

https://www.vmware.com/topics/glossary/content/network-traffic-analysis.html

https://www.rapid7.com/fundamentals/network-traffic-analysis/

https://www.forbes.com/advisor/business/software/what-is-pcap/

https://infocenter.nokia.com/public/7705SAR234R1A/index.jsp?topic=%2Fcom.nokia.oam-guide%2Fpcap_file_forma-ai9o99jjsa.html

https://www.endace.com/learn/what-is-a-pcap-file

https://www.wireshark.org/

https://en.wikipedia.org/wiki/Wireshark