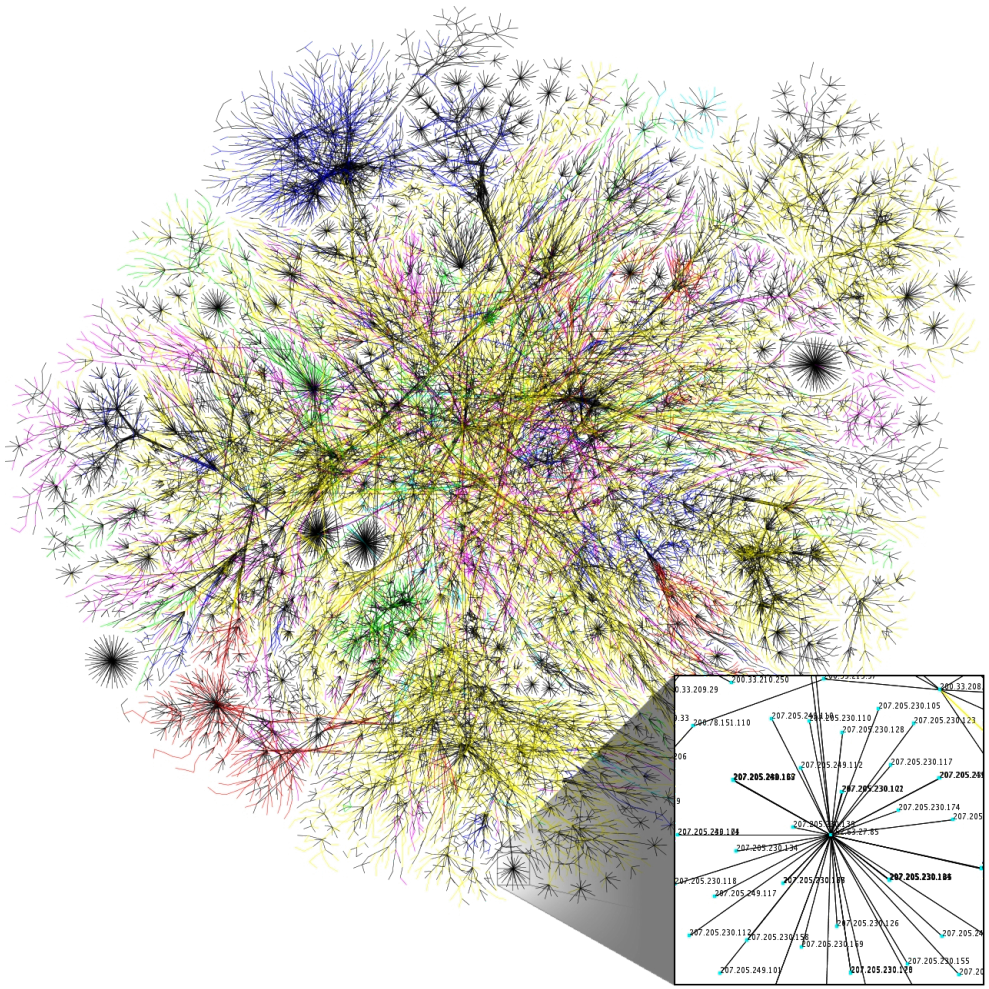
Networking

Vad är internet?

Internet


207.205.230.105
207.205.230.110
207.205.230.123
207.205.230.128
207.205.230.117
207.205.230.102
207.205.230.174
207.205.230.134
207.205.230.118
207.205.230.186
207.205.230.126
207.205.230.155

Internetprotokollet (IP)

IP over Avian Carriers, 1 April 1990

A Standard for the Transmission of IP Datagrams on Avian Carriers

Status of this Memo

This memo describes an experimental method for the encapsulation of IP datagrams in avian carriers. This specification is primarily useful in Metropolitan Area Networks. This is an experimental, not recommended standard. Distribution of this memo is unlimited.

Internetprotokollet (IP)

- Ett paket i IP kallas **datagram**.
- Består av en header och själva datan.

IP-headern

Headern innehåller bland annat:

- Datagrammets storlek
- Avsändare och mottagare (IP-addresser)
- *Time to live* (varför?)
- *Differentiated Services Code Point*, typ av tjänst (t.ex. VoIP)
- Checksumma

IPv4 vs. IPv6

Det finns olika versioner av IP:

- **IPv4**, 32 bitar, ca. 4 miljarder addresser (t.ex. `172.31.255.255`)
- **IPv6**, 128 bitar, löjligt många adresser (t.ex. `2001:db8::ff00:42:8329`)

340 282 366 920 938 463 463 374 607 431 768 211 456

Open Systems Interconnection Model (OSI)

IP är ett av flera skikt i **OSI-modellen**:

1. **Det fysiska skiktet**: kablar, spänningsnivåer, ettor och nollor
2. **Datalänkskiktet**: kommunikation mellan enheter: wifi, ethernet
3. **Nätverksskiktet**: `IP`, addressering
4. **Transportskiktet**: `TCP`/`UDP` (*fortsättning följer...*)
5. Ytterligare skikt, t.ex. `HTTP`, `SSL`...

Vad finns det för begränsningar med IP?

Begränsningar med IP

- Paket kan försvinna (best effort delivery)
- Paket kan dyka upp i fel ordning
- Fel kan uppstå på grund av t.ex. störningar

Lösning: TCP!

TCP (Transmission Control Protocol)

- Byggt ovanpå IP
- Förbindelseorienterat
- Försäkrar (typ) intakt data i rätt ordning
- Dubbelriktat
- Data delas upp i **segment** som skickas i IP-paket
- Segment numreras med **sekvensnummer**
- **ACK-segment** används för att säkerställa att paket kommer fram

The Two Generals' Problem

Veckans läxa

Veckans läxa

Utforma ett nätverksprotokoll för att spela mot varandra med era schackspel.

- Vem är “källan till sanning”?
- Hur hanterar ni saknade features (en passant etc.)?
- Samla specifikation i ett repo.
- Kanske även lite Rust-kod?

TCP i Rust

Två viktiga `structs`:

- TcpListener: Används för att ta emot anslutningar på serversidan.
- TcpStream: Används på båda sidor för att läsa och skriva data.

Klientsidan

```rust
use std::io::prelude::*;
use std::net::TcpStream;

fn main() -> std::io::Result<()> {
    let mut stream = TcpStream::connect("127.0.0.1:34254")?;

    stream.write(&[1])?;
    stream.read(&mut [0; 128])?;
    Ok(())
} // the stream is closed here
```

Serversidan

```
use std::net::TcpListener;

let listener = TcpListener::bind("127.0.0.1:8080").unwrap();
match listener.accept() {
    Ok((_socket, addr)) => println!("new client: {addr:?}"),
    Err(e) => println!("couldn't get client: {e:?}"),
}
```

Vad finns det för problem med detta?

```
// Funktion som körs varje frame
fn update(self: &mut Self) {
  // Vänta på meddelande från motståndare
  let msg = read_message_from_tcp_stream();

  // Uppdatera brädet
  self.update_board(&msg);
}
```

Lösning: `set_nonblocking()`

```
let mut stream = TcpStream::connect("127.0.0.1:7878")?;
stream.set_nonblocking(true)?;
```

Blockar inte, utan returnerar `WouldBlock`

```
match stream.read(&mut msg_buf) {
    Ok(_) => {},
    Err(ref e) if e.kind() == io::ErrorKind::WouldBlock => {},
    Err(e) => panic!("IO error: {e}"),
};
```