nst x ru
devclub.

Design of the Internet: Architecture Masterclass

Packets, DNS, CDNs, Protocols, and Fault Tolerance

DevClub @ NST | A Deep Dive into How the Internet Really Works

Today's Journey Through Internet Architecture

01

Packets & Protocols

TCP, UDP, and the magic of data transfer

02

DNS Resolution

The internet's phonebook explained

03

CDNs & Fault Tolerance

Building systems that never break

04

Hands-On Tools

Practical demonstrations you can try

05

Key Takeaways

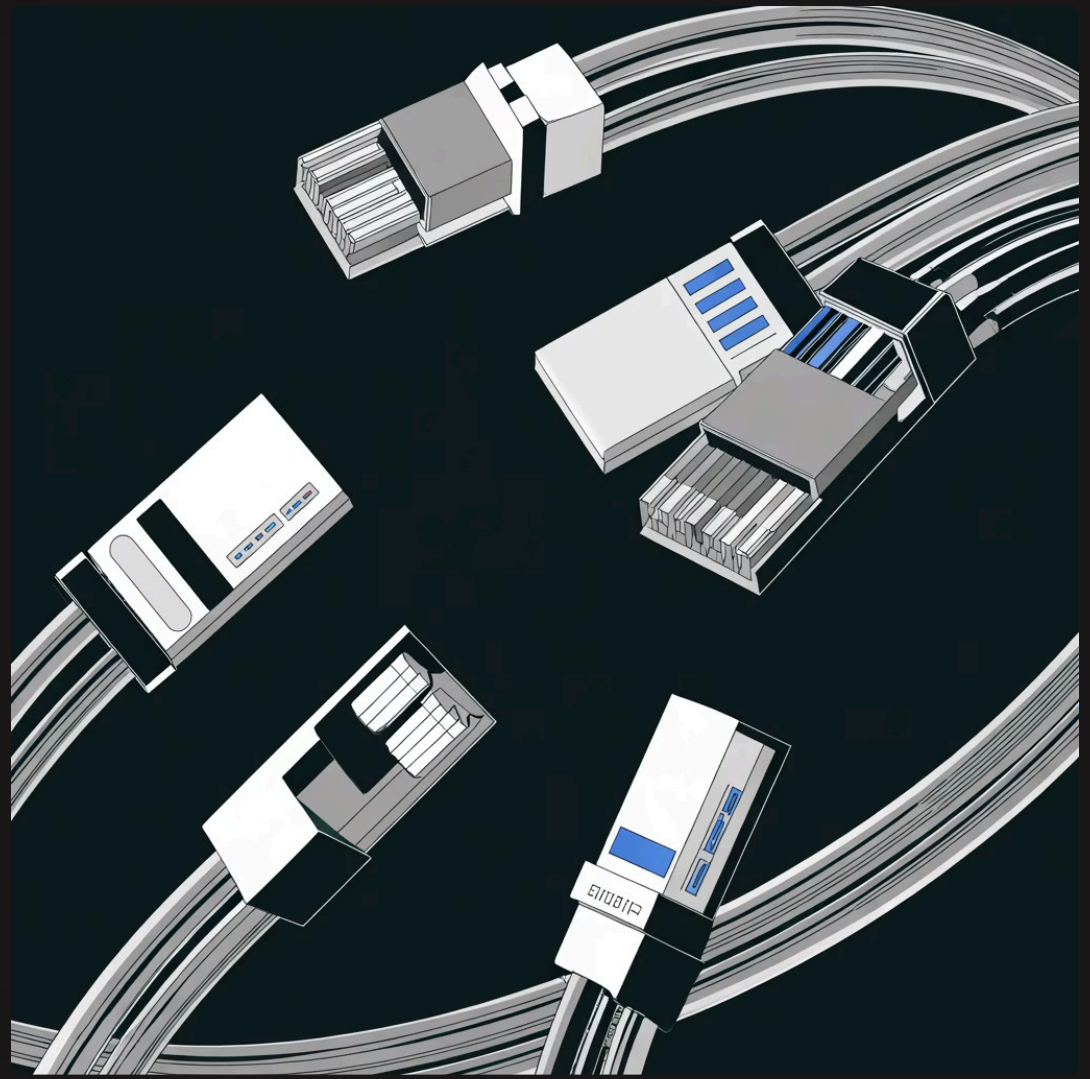
Apply these concepts to your projects

Today, we'll demystify the internet's backbone—knowledge that's vital for your hackathon wins and professional development. Understanding these foundational concepts will transform how you build and debug networked applications.

Packets: The Building Blocks of Internet Communication

Every time you load a webpage, stream a video, or send a message, your data doesn't travel in one piece. Instead, it's broken into thousands of tiny packets that journey independently across the internet, reassembling at their destination like a digital jigsaw puzzle.

But why? Breaking data into packets allows for efficient routing, error correction, and simultaneous multi-path transmission. If one packet fails, only that piece needs resending—not your entire file.



Inside a Packet: Anatomy of Internet Data

1

Header

Source & Destination IP: Where it's from and where it's going

Sequence Number: Order for reassembly

Protocol Info: TCP or UDP instructions

2

Payload

Actual Data: Your email text, video chunk, or website code

Size: Typically 1,500 bytes (MTU limit)

3

Footer

Checksum: Error detection code

Trailer: Marks packet end

Think of It Like Mailing a Book

Imagine mailing a 300-page book. You wouldn't send it as one massive, unwieldy package. Instead, you'd mail each page separately in its own envelope, numbered for reassembly.

If envelope #47 gets lost? Just resend that page—not the entire book. That's packet switching in action, and it's why the internet is remarkably efficient even when connections are imperfect.

Protocols: TCP vs. UDP

Not all data needs the same delivery guarantees. Choosing between TCP and UDP is like choosing between registered mail and regular post—each serves different purposes with distinct trade-offs.

Feature	TCP (Transmission Control Protocol)	UDP (User Datagram Protocol)
Reliability	High—acknowledgements and retransmission guarantee delivery	Low—fire and forget, no delivery confirmation
Speed	Slower due to overhead from error checking and ordering	Faster with minimal overhead, immediate transmission
Connection	Connection-oriented—requires handshake before data transfer	Connectionless—starts sending immediately
Use Cases	Web browsing, email, file transfers, APIs	Live streaming, gaming, video calls, DNS queries
Error Handling	Automatic retransmission of lost packets	Application must handle any errors

TCP: The Reliable Choice

Perfect when every byte matters—financial transactions, database operations, downloading files

UDP: The Speed Demon

Ideal when speed trumps perfection—a few dropped video frames won't ruin your Netflix experience

The TCP Three-Way Handshake

SYN (Synchronise)

Client sends SYN packet: "Hello server, I want to connect. Here's my sequence number."

SYN-ACK (Synchronise-Acknowledge)

Server responds with SYN-ACK: "I acknowledge your request. Here's my sequence number. Let's connect."

ACK (Acknowledge)

Client sends final ACK: "Acknowledged! Connection established. Let's transfer data."

This connection-oriented magic ensures both sides are ready before data flows. It adds latency but guarantees reliability—crucial for applications that can't tolerate data loss.

Real-World Impact

Every HTTPS request begins with this handshake. On high-latency connections (like satellite internet), this three-step process can add noticeable delay—which is why connection reuse and HTTP/2 multiplexing matter so much for web performance.

DNS: The Internet's Phonebook

Humans love memorable names like google.com or github.com. Computers need numerical IP addresses like 172.217.14.206. The Domain Name System (DNS) is the critical translation layer that makes the internet usable for humans whilst maintaining the efficiency computers need.

4.7B

DNS Queries

Processed globally every second

<50ms

Average Resolution

Time from query to response

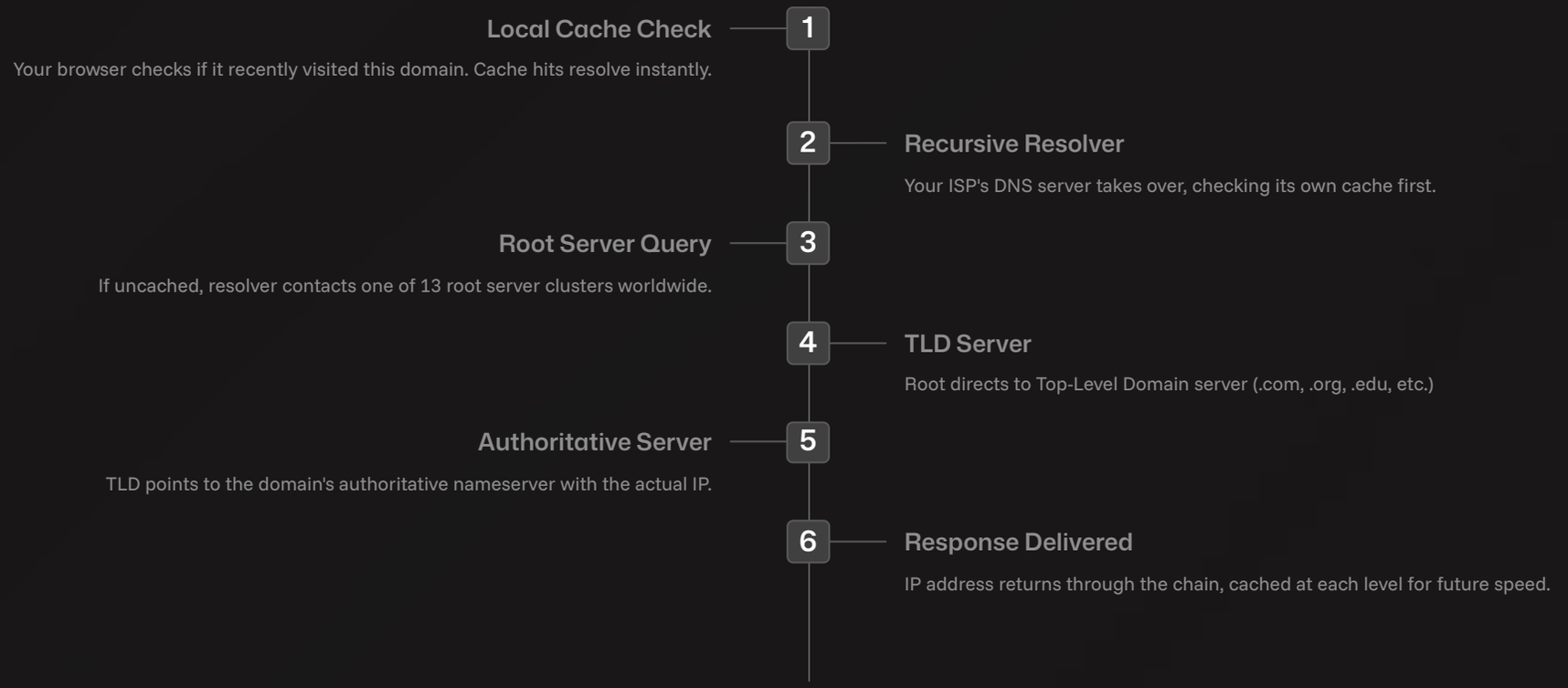
333M

Domain Names

Registered worldwide as of 2024

Without DNS, you'd need to memorise IP addresses for every website you visit. DNS operates invisibly, resolving billions of queries per second across a globally distributed hierarchy that's remarkably resilient to failure.

How DNS Resolution Actually Works



Key DNS Record Types

Type	Purpose	Example
A	Maps domain to IPv4	google.com → 172.217.14.206
AAAA	Maps domain to IPv6	google.com → 2607:f8b0::200e
CNAME	Creates domain alias	www.site.com → site.com
MX	Mail server routing	Routes email for domain
TXT	Arbitrary text data	SPF records, verification

TTL and Caching

Every DNS record includes a Time-To-Live (TTL) value—typically 300 to 86,400 seconds. This tells resolvers how long to cache the answer before checking again.

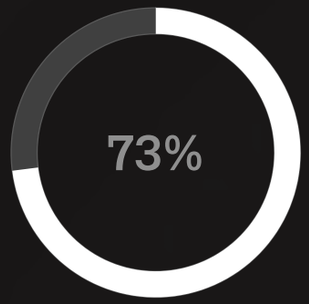
Trade-off: Longer TTLs mean faster lookups but slower propagation when you change servers. Shorter TTLs enable quick updates but generate more DNS traffic.

CDNs and Fault Tolerance: Building Unbreakable Systems

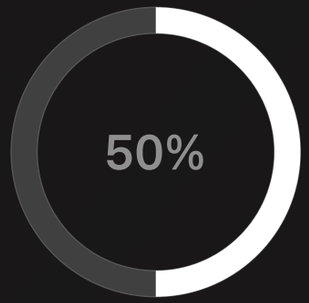
What is a Content Delivery Network?

A CDN is a globally distributed network of servers that cache and deliver content from locations closest to users. Instead of every request travelling to a single origin server—potentially thousands of miles away—CDNs serve content from nearby edge servers.

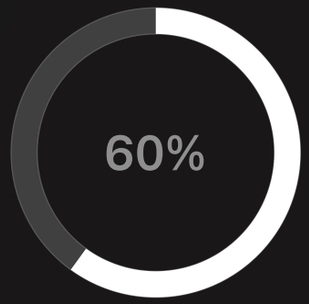
The result? Dramatically faster load times, reduced bandwidth costs, and resilience against server failures and traffic spikes.



Of internet traffic served through CDNs globally

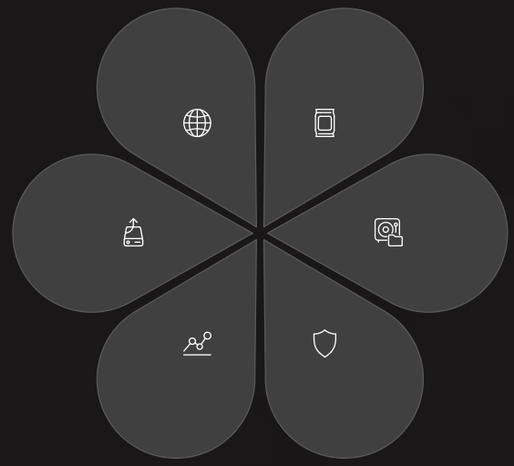


Average reduction in page load time



Bandwidth savings for origin servers

How CDNs Integrate with Everything



DNS Integration

CDNs use DNS to route users to nearest edge server via geolocation

Protocol Optimisation

CDNs maintain persistent TCP connections, reducing handshake overhead

Caching Strategy

Static assets cached; dynamic content accelerated via route optimisation

DDoS Protection

Distributed architecture absorbs attack traffic before reaching origin

Anycast Routing

Single IP address routes to multiple servers; closest one responds

Automatic Failover

If edge server fails, requests instantly reroute to next nearest node

Your Action Plan: Tools and Takeaways

Packets Are Efficient

Breaking data into small pieces enables error correction, parallel routing, and resilient transmission across imperfect networks

Protocol Choice Matters

TCP guarantees delivery for critical data; UDP prioritises speed for real-time applications—choose wisely for your use case

DNS Powers Everything

Understanding DNS hierarchy and caching helps you debug connectivity issues and optimise application performance

CDNs Enable Scale

Global content distribution isn't optional—it's essential for speed, reliability, and handling traffic spikes in production systems

Hands-On Practice

Theory means nothing without practice—use the tools below to explore these concepts in real-world scenarios

Essential Tools to Try This Week

Wireshark

Capture and analyse packets in real-time. See TCP handshakes, packet headers, and protocol details visually.

dig Command

Query DNS records directly. Run `dig google.com` to trace the full resolution process and inspect TTL values.

Browser DevTools

Inspect network timing, view CDN headers, and analyse waterfall charts to understand real-world performance.

Next Steps: Apply these concepts to your next hackathon project. Implement proper error handling for network failures, choose appropriate protocols, and consider CDN integration for any public-facing application. The internet's architecture isn't just theory—it's the foundation of everything you'll build.