
📡 Network Protocols in System Design

---

1. What are Network Protocols?

- A **network protocol** is a set of **rules and conventions** for communication between devices in a network.
- They define **how data is formatted, transmitted, routed, and received**.
- Without protocols, two systems would not “understand” each other.

In **system design**, protocols are fundamental because: [servers, regions, data centers).

- Every interaction (API call, DB query, file download) is governed by one or more protocols.

---

2. OSI Model & Protocols

The **OSI (Open Systems Interconnection)** model helps us categorize protocols.

Layer	Purpose	Example Protocols	System Design Relevance
7. Application	Defines how apps use the network	HTTP/HTTPS, gRPC, WebSocket, SMTP, DNS	Web APIs, microservices, DNS lookups
6. Presentation	Data encoding, encryption	TLS/SSL, JSON, Protobuf	Security, serialization
5. Session	Manage sessions & connections	RPC, NetBIOS	Persistent sessions
4. Transport	Reliable or fast delivery	TCP, UDP, QUIC	API calls, video streaming
3. Network	Routing packets	IP, ICMP, BGP	Internet routing
2. Data Link	Error detection, frames	Ethernet, ARP, PPP	LAN, switches
1. Physical	Transmission medium	Wi-Fi, Fiber, 5G	Hardware layer

👉 In **system design**, we mostly deal with **layers 4–7**, but lower layers still matter (e.g., Wi-Fi vs mobile data can impact latency).

---

3. Key Categories of Protocols in System Design

🔹 3.1 Communication Protocols (Application Layer)

These define **how clients, servers, and services talk**.

- **HTTP/HTTPS (HyperText Transfer Protocol)**
    - Foundation of the web.
    - Stateless, request-response.
    - HTTPS = HTTP + TLS encryption.
    - Used for APIs (REST, GraphQL).
- **gRPC**
    - High-performance RPC framework.
    - Uses HTTP/2, binary (Protobuf).
    - Popular for **microservices**.
- **WebSockets**
    - Full-duplex (two-way) communication.
    - Used for **real-time apps** (chat, live updates, gaming).
- **MQTT (Message Queuing Telemetry Transport)**
    - Lightweight publish-subscribe.
    - Optimized for **IoT devices**.
- **DNS (Domain Name System)**
    - Translates **domain → IP address**.

    - First step before any network request.

🔹 3.2 Transport Protocols

Define **how data packets travel** between endpoints.

- **TCP (Transmission Control Protocol)**
    - Reliable, ordered, connection-oriented.
    - Ensures no data loss (retransmission, ACKs).
    - Used in: HTTP, gRPC, SSH, database connections.
    - **Trade-off**: Slower due to handshakes.
- **UDP (User Datagram Protocol)**
    - Connectionless, faster, no guarantee of delivery.
    - Used in: video calls, gaming, live streaming.
    - **Trade-off**: Data may be lost but latency is low.
- **QUIC (Quick UDP Internet Connections)**
    - Google's protocol (used in HTTP/3).
    - Combines UDP's speed + TCP-like reliability.
    - Better for mobile networks (fewer drops).

🔹 3.3 Security Protocols

Protect data **in transit**.

- **TLS/SSL** → Encrypts communication (HTTPS).
- **SSH (Secure Shell)** → Remote login & tunneling.
- **IPSec (Internet Protocol Security)** → Used in VPNs.
- **Kerberos / OAuth / SAML** → Authentication & identity management in distributed systems.

🔹 3.4 Data Routing Protocols

Ensure **packets find the right path**.

- **IP (Internet Protocol)** → Assigns unique addresses, routes packets.
- **BGP (Border Gateway Protocol)** → Decides inter-network routing (between ISPs).
- **OSPF (Open Shortest Path First)** → Intra-network routing inside organizations.
- **ICMP (Internet Control Message Protocol)** → Used for diagnostics (ping, traceroute).

🔹 3.5 Data Storage & Messaging Protocols

- **Kafka Protocol** → Custom binary protocol over TCP for event streaming.
- **AMQP (Advanced Message Queuing Protocol)** → Messaging systems (RabbitMQ).
- **Redis/Memcached Protocols** → In-memory cache communication.
- **SQL/NoSQL Protocols** → Proprietary protocols (e.g., PostgreSQL wire protocol, MongoDB BSON).

---

4. Protocols in Real System Design Examples

Example 1: E-commerce Website

- User enters URL → **DNS** resolves domain.
- Frontend → Backend → **HTTPS (TCP+TLS)**.
- Microservices talk → **gRPC over HTTP/2**.
- Payment → **HTTPS + OAuth 2.0**.
- Notification → **WebSockets** (order updates).
- Database queries → **TCP (Postgres wire protocol)**.

Example 2: Chat Application

- Login → **HTTPS + TLS**.
- Messaging → **WebSockets (TCP)** for real-time.

- File sharing → **HTTPS file upload**.
- Notifications → **MQTT** (for mobile push).
- Video calls → **RTP over UDP** (low latency).

Example 3: Video Streaming (e.g., YouTube, Netflix)

- Content Delivery → **HTTP/3 (QUIC)**.
- Video chunks → **DASH/HLS (built on HTTP)**.
- Realtime comments → **WebSockets**.
- Ads & Tracking → **HTTPS** APIs.

---

5. Trade-offs in Choosing Protocols

When designing, engineers must choose based on **system requirements**:

Protocol Choice	Best For	Trade-offs
TCP	Reliable, ordered delivery	Slower due to handshakes
UDP	Real-time, low latency	No reliability
HTTP	Universal, simple APIs	Verbose, slower than binary
gRPC	Fast microservices	Harder debugging

WebSockets	Real-time apps	More complex state mgmt
MQTT	IoT, constrained devices	Not good for large-scale web
QUIC	Mobile, fast web	Not yet as widely supported

---

6. Diagram (How Protocols Fit in a System)

Imagine a **multi-tier web app**:

```
[Client Browser]
  | DNS (UDP/TCP)
  v
[CDN / Load Balancer]
  | HTTP/HTTPS (TCP/QUIC)
  v
[API Gateway]
  | gRPC / REST (HTTP/2 over TCP)
  v
[Microservices]
  | TCP (DB protocols, Redis)
  v
[Databases, Caches, Queues]
```

- Security: **TLS/SSL** at multiple points.
- Monitoring: **ICMP** for health checks.
- Real-time updates: **WebSockets** alongside HTTP.