Week 2: Application Layer

NOS_Lecture_slides_WK02.pdf

Application Layer Overview

The application layer is the topmost layer in the OSI model, directly serving end-user applications.

TCP/IP Model

- Application Layer - Encodes/decodes the message in a form that is understood by the sender and the recipient.
- Transport Layer - breaks down the message into packets. Each packet is numbers and within the total packets. Which then is sent to the recipient to assemble the packets together correctly.
- Network Layer - adds the sender's and that of the recipient. The network then knows where to send the message, and where it came from.
- Link Layer - enables the transfers of packets between nodes on a network. Also between different networks.

TCP Services:

- Reliable transport
- Flow Control
- Congestion Control
- does not provide timing
- connection-oriented

UDP Services:

- Unreliable data transfer
- does not provide reliability

Key Protocols

- **HTTP (Hypertext Transfer Protocol)**- Used for web browsing and data transfer- Request-response protocol between client and server- Supports methods like GET, POST, PUT, DELETE
- **FTP (File Transfer Protocol)**- Specialized for file transfer between systems- Uses separate control and data connections- Supports authentication and file operations
- **SMTP (Simple Mail Transfer Protocol)**- Email transmission protocol- Push protocol for sending messages- Works with POP3/IMAP for complete email service
- **DNS (Domain Name System)**- Translates domain names to IP addresses- Hierarchical naming structure- Distributed database system

TLD - Top-Level Domain servers → responsible for addresses like .com, .org, .net, .edu, .aero and countries like .uk, .fr for a better IP mappings.

Application Layer Overview

- **Principles of Network Applications:** Web and HTTP, Email (SMTP, IMAP), DNS, P2P applications, Video streaming, and content distribution networks.
- **Client-Server Paradigm:** Server is always-on with a permanent IP, clients connect intermittently.
- **P2P Architecture:** No always-on server, peers communicate directly.

Processes Communicating

- **Sockets:** Used for sending/receiving messages between processes.
- **Addressing Processes:** Uses IP address and port numbers.

HTTP

- **HTTP Overview:** Stateless protocol using TCP, with non-persistent and persistent connections.
- **HTTP Methods:** GET, POST, HEAD, PUT.
- **HTTP Response Codes:** 200 OK, 301 Moved Permanently, 400 Bad Request, 404 Not Found.

Email

- **Components:** User agents, mail servers, SMTP.
- **Mail Access Protocols:** IMAP, HTTP.

DNS

- **Services:** Hostname to IP translation, host aliasing, mail server aliasing, load distribution.
- **Structure:** Distributed, hierarchical database with root, TLD, and authoritative servers.
- **DNS Records:** Types A, CNAME, NS, MX.

P2P Applications

- **File Distribution:** Comparison between client-server and P2P.
- **BitTorrent:** File divided into chunks, peers exchange chunks.

Video Streaming and CDNs

- **Challenges:** Bandwidth variability, packet loss, client interactivity.
- **DASH:** Dynamic Adaptive Streaming over HTTP.
- **CDNs:** Distribute content across multiple servers to handle large-scale streaming.

Socket Programming

- **UDP and TCP Sockets:** Building client/server applications using sockets.
- **Example Applications:** Python code for UDP and TCP clients and servers.

Client-Server Architecture

- Server provides services and resources
- Client requests and consumes services
- Communication through standardized protocols

Application Layer Services

- Network Security- Authentication and authorization- Data encryption- Digital certificates
- Resource Sharing- File and printer sharing- Database access- Remote procedure calls

API and Interface Design

- Application Programming Interfaces (APIs)
- RESTful services
- SOAP protocols
- Interface documentation standards

Common Application Layer Issues

- Performance Challenges- Latency and bandwidth limitations- Server overload- Network congestion
- Security Concerns- DDoS attacks- Man-in-the-middle attacks- Data breaches

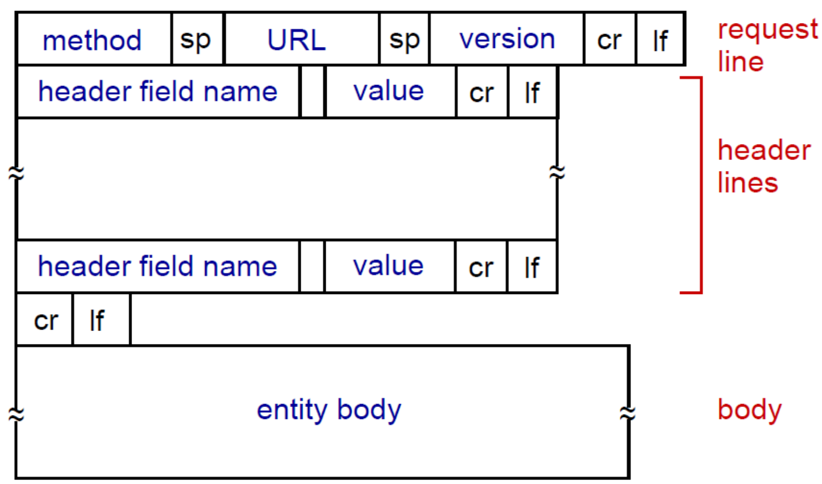
Best Practices

1. Implement proper error handling
2. Use secure protocols (HTTPS, SFTP)
3. Regular security audits
4. Performance monitoring
5. Load balancing implementation

Future Trends

- Microservices architecture
- Cloud-native applications
- API-first design
- Container orchestration

HTTP request message: general format



Caching example

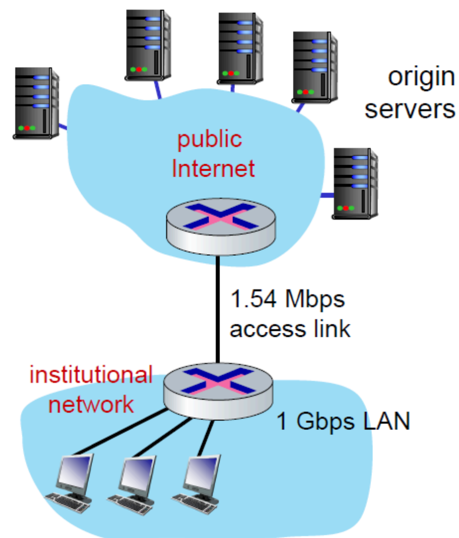
Scenario:

- access link rate: 1.54 Mbps
- RTT from institutional router to server: 2 sec
- Web object size: 100K bits
- Average request rate from browsers to origin servers: 15/sec
  - average data rate to browsers: 1.50 Mbps

Performance:

- LAN utilization: .0015
- access link utilization = .97
- end-end delay = Internet delay + access link delay + LAN delay = 2 sec + minutes + usecs

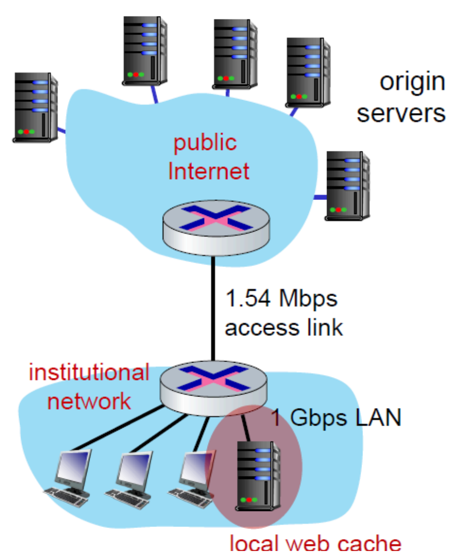
problem: large delays at high utilization!



Caching example: install a web cache

Calculating access link utilization, end-end delay with cache:

- suppose cache hit rate is 0.4: 40% requests satisfied at cache, 60% requests satisfied at origin
- access link: 60% of requests use access link
- data rate to browsers over access link
  = 0.6 * 1.50 Mbps = .9 Mbps
- utilization = 0.9/1.54 = .58
- average end-end delay
  = 0.6 * (delay from origin servers)
  + 0.4 * (delay when satisfied at cache)
  = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs



Summary

- **Application Architectures:** Client-server, P2P.
- **Service Requirements:** Reliability, bandwidth, delay.
- **Transport Services:** TCP (reliable), UDP (unreliable).

- **Protocols:** HTTP, SMTP, IMAP, DNS, BitTorrent.
- **Video Streaming and CDNs:** Handling large-scale content distribution.
- **Socket Programming:** Practical implementation using TCP and UDP.