Week 02 · Teacher guide

Before class

Open the animation and slides; run the local chat in two windows. Test Send, Pause polling, Resume and Mark read. Print one worksheet per pair and cut one set of packet cards. Prepare a projector and a timer. Students need no account; laptops are optional. Plan 75 minutes. The syllabus places week 2 on 28 November 2026.

Learning outcomes

Students should trace a message through clients, DNS, routers and a relay server; distinguish a name from an address; explain why ordering and recovery matter; and state what accepted, delivered and read each mean.

Materials and preparation

Teacher computer and projector; Python 3 for the real chat; paper, pens and scissors for the human network. No third-party Python packages are needed to run the class. Open dist/index.html directly if the server is unavailable. The worksheet and all PDFs are available offline. External readings are optional.

Teaching rhythm

Ask for a prediction, give quiet thinking time, compare in pairs, run one step, then ask for an explanation. Keep the animation paused whenever you ask a question. Use the same phrase throughout: What can the sender know now? In a shorter session, omit optional reading and reduce the paper activity; preserve the real chat and exit check.

00–03 · Slide 1 · What happens when you press Send?

Open index.html on Journey. Ask for 20 seconds of silent prediction, then two suggestions. Do not reveal the route yet. Link to week 1: the message becomes bytes, but bytes still need an agreed route and protocol. Today students will trace a message and distinguish server acceptance, delivery and reading.

03–06 · Slide 2 · Last week: bits need a rule

Expected: encoding, format and agreed protocol. Ask what ordinary volatile RAM loses on power failure. This prepares students for the local chat: its messages live in server memory and disappear when the process restarts. Do not suggest that every chat service has this storage policy.

06–10 · Slide 3 · Client and server

Expected: no, our relay design sends through the chat server. Client and server describe roles, not a device's size. A laptop can run both. Draw Alice → server → Ben. This is one architecture; peer-to-peer applications also exist. Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview

10–14 · Slide 4 · Name, address, port, path

Answer: /chat/ is the path. A port identifies a transport endpoint/service on a host; the URL path identifies a resource handled by the HTTP service. localhost refers to the current computer. The local demo is HTTP on loopback; real public chat should use HTTPS. IP addresses can be shared or change; they are not permanent person IDs. Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview

14–19 · Slide 5 · DNS finds an address

Open DNS lab. Look up twice, then expire the cache. First lookup is modeled as a resolver finding an answer; the second uses cached data. Answer: no, a valid cache entry or an existing connection can avoid a new lookup. chat.example and 203.0.113.10 are documentation examples, not a real chat host. The lab compresses a distributed lookup into one step. Source: https://www.cloudflare.com/learning/dns/what-is-dns/

19–23 · Slide 6 · A message becomes network data

Use the packet lab. The three chunks are a teaching model, not measured IP packets. Real packetization depends on the protocol, network limits and implementation. TCP is a byte stream: it does not preserve application message boundaries. An application needs framing, such as a structured HTTP body. Avoid saying every packet has a whole-message number and total-count field. Source: https://www.rfc-editor.org/rfc/rfc9293.html

23–27 · Slide 7 · Routers forward packets

Step through Journey, then select alternate route and reset. Explain that the animation deliberately simplifies routing. Different routes are possible; do not imply every packet normally takes a different one. Wi-Fi is one local link, not the whole internet. Our local chat's loopback traffic does not actually cross these routers.

27–32 · Slide 8 · Arrival order can differ

Open Packets. Click packet 3, then 1, then 2. Ask for the resulting message before showing all chunks. Expected: sequence information and known message framing. The lab labels chunks 1–3 for readability. TCP uses byte sequence numbers, not these classroom chunk numbers. Source: https://www.rfc-editor.org/rfc/rfc9293.html

32–37 · Slide 9 · Missing data needs recovery

Reset packets, choose Drop chunk 2, then deliver all. Chunk 2 is marked lost; the full message stays hidden. Use Retry missing to complete it. Explain that our button stands in for transport recovery. TCP acknowledgments concern received bytes, not whether a person read them. UDP alone does not provide TCP's ordered reliable stream; applications may add reliability. Source: https://www.rfc-editor.org/rfc/rfc9293.html

37–41 · Slide 10 · Accepted is not read

Use Receipts. Accept, attempt delivery with recipient offline, bring recipient online, deliver, then mark read. Answer: yes in this demo, which queues in memory. Read receipts are application conventions and do not prove understanding. Real apps differ in their receipt rules. Keep transport ACK, HTTP response, delivery receipt and read receipt separate.

41–45 · Slide 11 · Trace a real local chat

Start START-CLASS.command before class. Open /chat/?user=Alice and /chat/?user=Ben in two windows. Send a fictional message. Point out the server event log and each browser's local request log. This is real HTTP via localhost, with polling every second. There is no real DNS lookup or internet route in this demo. Sources: local chat/server.py and chat/index.html; https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview

45–50 · Slide 12 · Make the recipient go offline

Ask for prediction, then pause Ben, send, inspect Accepted, resume, inspect Delivered, and press Mark read on Ben. Pause means this client stops fetching; it is a controlled model of disconnection. The server still runs. Status labels come from explicit application endpoints. Reading is manually reported, not inferred from fetching. Reloading preserves server data; restarting the server clears it.

50–55 · Slide 13 · Inspect one request

Answer: in the server's in-memory message list. Open browser developer tools > Network if comfortable; otherwise use the built-in log. Read the JSON fields id, sender, recipient, text, created_at and status. Explain that the browser supplies input but the server checks allowed names and message length. This is a teaching app without authentication, not a production service.

55–60 · Slide 14 · Be the network

Use packet cards in the worksheet. Give the sender an address only after asking DNS. Route paper packets to the server, then recipient; ask the class to watch for order and missing pieces. DNS gives an address and leaves the message path. This is an analogy spanning multiple layers, not an exact TCP simulation. No-device fallback for the real chat: students act out accepted, delivered and read.

60–64 · Slide 15 · Find the failure

Use Journey DNS failure and Server unavailable scenarios. Expected: DNS failure need not mean Wi-Fi is broken; server unavailable need not mean DNS failed; recipient offline can coexist with server acceptance. Existing connections and caches complicate real diagnosis. Ask for evidence before claiming the whole internet is down. If time is short, do one scenario only.

64–68 · Slide 16 · Rebuild the journey

Blank the screen for 30 seconds. Expected: valid DNS answers and established connections can be reused; the receiving application interprets encoded data. Have partners name client/server roles, one possible failure, and what a receipt proves. Sequence is a simplified first-send story; real protocols add handshakes, encryption and more details. Preview week 3: who can read the data?

68–71 · Slide 17 · Choose an exploration

Show Resources in the webpage and point to the local worksheet. External reading is optional; every classroom activity works without internet. Sources: https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Web_standards/How_the_web_works and https://www.rfc-editor.org/rfc/rfc9293.html . The RFC is teacher-depth reading, not a required student assignment.

71–73 · Slide 18 · Map a message's journey

Optional 10–15 minute home task. Success: DNS is beside the data path, addresses and names are distinguished, and accepted/delivered/read are not conflated. Coding route: server already supplies created_at; uncomment the timestamp display line in chat/index.html. Explain that it is server acceptance time, not read time. The worksheet contains the exact change and a check.

73–75 · Slide 19 · Before you go

Answers: 1 no, it supplies name-resolution information; 2 no, acceptance is a server/application event; 3 sequence information plus framing, with missing data recovered when the transport/application provides it. Collect answers to choose a brief recap next week. End at 75 minutes.

DNS and addresses

DNS is distributed name resolution. Resolvers may use cached answers until their lifetime expires. A name can resolve to multiple addresses, and addresses can change or be shared. Our lab has one example mapping and a manual expiry control. DNS does not relay the application message. localhost in the actual chat is a loopback name; this exercise does not produce a public DNS lookup.

Layers and packet boundaries

A user-visible message is an application concept. HTTP carries a structured request body; transport and IP handle other jobs. TCP provides an ordered byte stream with retransmission, but does not preserve application message boundaries. Our three chunks, labeled 1–3, are pedagogical pieces rather than real captured packets. Never claim that every IP packet contains a message number, chunk index and total chunk count. Routers forward packets using routing information; they do not need to understand the chat text.

Reliability has limits

TCP can recover missing data and restore byte order while a connection is viable. It can also time out or fail. UDP alone does not provide these ordered-stream guarantees. Some protocols built on UDP add reliability; HTTP/3 uses QUIC rather than TCP. Keep the main lesson on our local HTTP example and avoid teaching that all internet traffic is TCP.

Four kinds of acknowledgment

A TCP ACK concerns transport bytes. An HTTP success response means what the server endpoint defines: our POST returns 201 after adding the message to memory. A delivery receipt in our app occurs after the recipient retrieves the message and POSTs a delivered receipt. Read is a second explicit POST after pressing Mark read. None of these proves that a person understood the text. Actual products define their own status rules.

Polling, push and offline delivery

Our recipient polls once each second. Pausing polling leaves messages waiting in the server process, and resuming fetches them. Many chat products use persistent connections for lower-overhead push. This local demo is not a claim about every messaging architecture. If the server process stops, this app loses its in-memory list; a production system may use durable storage.

Boundaries of the real demo

Both clients and the server run on the teacher computer. Requests are real HTTP but stay on loopback. The toy service is unauthenticated, has two fixed identities, caps messages and checks basic request values. It is not intended for public deployment or private messages. HTTPS and who can read data are the bridge to week 3, not fully demonstrated here.

Troubleshooting

If the browser opens too early, reload after the terminal prints the local URL. If connection is refused, start python3 chat/server.py. If the port is in use, stop the older instance or use --port 8766 and update browser URLs manually. If the two windows show the same identity, check ?user=Alice and ?user=Ben. If status stays accepted, resume Ben. If it stays delivered, press Mark read in Ben. If Python is unavailable, use Receipts and the paper role-play.

Likely questions

Is Wi-Fi the internet? No: Wi-Fi is a local wireless link that may provide access to other networks. Is a server always a giant machine? No: server describes a role. Do all packets take different routes? No: they can, but often share a route. Does a sent check prove delivery? Only if that application defines it that way; inspect the actual status semantics. Can a read receipt prove attention? No.

Answer key

Warm-up: interpretation comes from encoding/format/context. URL: localhost is the name, 8765 is the port, /chat/ is the path. DNS: no new lookup is required for every message when an answer or connection can be reused. Packet activity: pieces 3,1,2 reconstruct Meet at 4; with piece 2 missing the message is incomplete. Receipt activity: the server may accept while Ben is paused; delivered follows Ben fetching and acknowledging; read follows the explicit read action. Exit: DNS does not forward the message; acceptance does not prove reading; sequence information and framing support reconstruction.

Homework success criteria

Accept paper or code. A good map puts DNS beside the message path, identifies sender and recipient clients and the relay server, labels request/response, and explains one failure without calling every failure an internet outage. For code, uncomment the provided timestamp line, reload the chat, send a new message and explain that the displayed time is the server acceptance time. Do not confuse it with delivery or reading time.

Sources

MDN — How the web works

https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Web_standards/How_the_web_works

MDN — HTTP overview

https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview

Cloudflare — What is DNS?

https://www.cloudflare.com/learning/dns/what-is-dns/

RFC 9293 — Transmission Control Protocol

https://www.rfc-editor.org/rfc/rfc9293.html