

Week 4: The Transport Layer

CSCI 4431/6431 Computer Networks

Prof. Tim Wood

George Washington University

timwood@gwu.edu

<https://go.gwu.edu/csnetworks>

Today

- **Quiz** on the Chapter 3 intro videos
- Where the transport layer sits, and how it finds your process
- **UDP**: what it includes and what it makes you build yourself
- Worksheet
- **Reliable data transfer** from scratch
- **Project 1**: pick a protocol, start your spec

Housekeeping

- Lab 1 and DNS "Quiz" — lots of problem solving
- **Project 1 teams** should be formed. Not on a team?
Form later in class.
- **Discord** for questions
- Everything at <https://go.gwu.edu/csnetworks>

Quiz

Four questions. ~8 minutes.

Covers videos **3.1** and **3.2** on socket demultiplexing.

Chapter 3 - Slides 4 - 9

- Transport gives **logical communication between processes**
- Network gives communication between **hosts**
- Ann and Bill's houses: who does what?
- TCP vs UDP — the one-slide version

Slides: https://gaia.cs.umass.edu/kurose_ross/ppt-9e/Chapter_3_v9.0.pptx

Chapter 3 - Slides 20 - 25

- Multiplexing at the sender, demultiplexing at the receiver
- **UDP:** demux on **destination port only**
- **TCP:** demux on the **4-tuple** — src IP, src port, dst IP, dst port

Quiz Q2 — how many sockets?

```
udp.dstport == 5000
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.14	10.0.0.7	UDP	73	54321 → 5000 Len=31
2	0.004120	192.168.10.22	10.0.0.7	UDP	73	54321 → 5000 Len=31
3	0.009755	172.16.4.9	10.0.0.7	UDP	80	61002 → 5000 Len=38
4	0.015002	192.168.10.14	10.0.0.7	UDP	66	54321 → 5000 Len=24
5	0.021447	172.16.4.9	10.0.0.7	UDP	80	61002 → 5000 Len=38
6	0.026918	192.168.10.22	10.0.0.7	UDP	73	54321 → 5000 Len=31

Quiz Q2 — how many sockets?

udp.dstport == 5000						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.14	10.0.0.7	UDP	73	54321 → 5000 Len=31
2	0.004120	192.168.10.22	10.0.0.7	UDP	73	54321 → 5000 Len=31
3	0.009755	172.16.4.9	10.0.0.7	UDP	80	61002 → 5000 Len=38
4	0.015002	192.168.10.14	10.0.0.7	UDP	66	54321 → 5000 Len=24
5	0.021447	172.16.4.9	10.0.0.7	UDP	80	61002 → 5000 Len=38
6	0.026918	192.168.10.22	10.0.0.7	UDP	73	54321 → 5000 Len=31

- *Three source hosts. One server socket.*

Quiz Q3 — This time with TCP

```
tcp.dstport == 80
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.14	10.0.0.7	TCP	74	9157 → 80 [SYN] Seq=0 Win=64240 Len=0
2	0.031885	172.16.4.9	10.0.0.7	TCP	74	5775 → 80 [SYN] Seq=0 Win=64240 Len=0
3	0.052013	172.16.4.9	10.0.0.7	TCP	74	9157 → 80 [SYN] Seq=0 Win=64240 Len=0
4	0.061204	192.168.10.14	10.0.0.7	TCP	466	9157 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=412
5	0.079330	172.16.4.9	10.0.0.7	TCP	471	5775 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=417
6	0.088741	172.16.4.9	10.0.0.7	TCP	463	9157 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=409

Quiz Q3 — This time with TCP

```
tcp.dstport == 80
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.14	10.0.0.7	TCP	74	9157 → 80 [SYN] Seq=0 Win=64240 Len=0
2	0.031885	172.16.4.9	10.0.0.7	TCP	74	5775 → 80 [SYN] Seq=0 Win=64240 Len=0
3	0.052013	172.16.4.9	10.0.0.7	TCP	74	9157 → 80 [SYN] Seq=0 Win=64240 Len=0
4	0.061204	192.168.10.14	10.0.0.7	TCP	466	9157 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=412
5	0.079330	172.16.4.9	10.0.0.7	TCP	471	5775 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=417
6	0.088741	172.16.4.9	10.0.0.7	TCP	463	9157 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=409

- *Three source hosts. Three server sockets.*

Connectionless Transport

UDP

Chapter 3 - Slides 27 - 33

- "No frills": segments may be **lost** or arrive **out of order**
- **Connectionless** — no handshake, no state, each segment on its own
- Who uses it: DNS, SNMP, streaming media, **HTTP/3**
- The segment header: 8 bytes, four fields, tiny!

The UDP header

Field	Bytes
Source port	2
Dest port	2
Length	2
Checksum	2

Length counts the header too. Length 40 → 32 bytes of data.

Compare: TCP's header is 20 bytes minimum.

Chapter 3 - Slides 34 - 38

- Checksum: add the 16-bit words, wrap the carry around, send the complement
- Receiver recomputes and compares
- **Detects** *some* errors. Does not **fix** them.

UDP Worksheet

Why does DNS run over UDP?

(most of the time)

- Turn in at the end of class
- Ask Esma and me for help!

Worksheet debrief

- Part 2: TCP costs **2× the latency** for a one-shot lookup
- Part 3: **33.6 Mbps** of pure overhead at 50k lookups/sec
- Part 4: **2,000 sockets vs. 1** ← this is the most critical
- Part 5: **too short a wait** → unnecessary retransmissions;
too long a wait → high delay to detect loss

What's in those 2,000 sockets?

If TCP is holding state for every client, what is it *holding*?

- Buffers?
- Sequence numbers?
- Timers?

These are things we need to ensure reliable transfer.

Break (~4:42pm?)

Principles of Reliable Data Transfer

How do you build reliable transfer on top of UDP's best effort service?

Chapter 3 - Slides 40 - 45

- A **reliable channel** is an abstraction the app gets to assume
- Underneath it: an unreliable channel that loses and corrupts
- Sender and receiver **cannot see each other's state**
 - the only way to learn anything is to be told
- We will specify protocols with **finite state machines**

Chapter 3 - Slides 46 - 48

- **rdt1.0** — perfect channel. Sender sends, receiver receives. Done.
- **rdt2.0** — the channel flips bits.
 - checksum to **detect**
 - **ACK / NAK** to report
 - retransmit on NAK
- **stop and wait**: one packet out, then wait

Chapter 3 - Slides 49 - 53

- rdt2.0 as an FSM, sender and receiver
- Walk the no-error case, then the corrupted-packet case
- **The fatal flaw:** what if the *ACK* is corrupted?
 - sender can't tell what happened
 - can't just retransmit — that makes a **duplicate**

Chapter 3 - Slides 54 - 56

- **rdt2.1** — sequence numbers. Two values (0 and 1) are enough. *Why?*
- Twice as many states: the sender must remember which one it's on
- The receiver must detect and discard duplicates
- **rdt2.2** — drop NAKs entirely; ACK the last good packet instead
 - this is what TCP actually does (*see book / slides for details*)

Project 1

Writing a specification

Project 1 — what happens today

- With your team, **pick a protocol** to design
- At least two teams per protocol
- Start writing **Spec v1**, *without* talking to your partner team

Spec v1 is due before class next week.

Details: <https://go.gwu.edu/csnetworks/>

Your spec has to define

Chapter 2, slide 11 — the same four things, for *your* protocol:

- **Message types** — what messages exist
- **Message syntax** — the fields, and how they are delimited
- **Message semantics** — what each field means
- **Rules** — when a message may be sent, and what must be answered

Use real RFCs as a guide! (But doesn't need to be ASCII)

Everything runs over UDP

But for simplicity we will assume **the network is reliable!**

What specs Look like

- **RFC 768 — User Datagram Protocol (UDP):**
datatracker.ietf.org/doc/html/rfc768 — minimal example, ~2.5 pages
- **RFC 1939 — Post Office Protocol version 3 (POP3):**
datatracker.ietf.org/doc/html/rfc1939 — richer example, ~19-20 pages

Before you leave today

1. Team has a **protocol** picked
2. Team has a **way to reach each other** and a time to meet
3. At least a rough list of your **message types**

*The rest is homework. Spec v1 due **next class**.*

Wrap up and Homework

- **Before next week:**
 - [Project 1](#) — Spec v1 due 9/23
 - Watch [3.4a](#) and [3.4b](#) — reliable data transfer, the rest of today's last block
 - Watch [3.5a](#) — TCP intro
- **Next week:** TCP, quiz, and Project 1 Phase 2 — working with your partner team

Chapter 3 Resources

- Textbook Slides
- Videos (Full playlist):
 - 3.1 Transport Layer Services
 - 3.2 Multiplexing and Demultiplexing
 - 3.3 Connectionless Transport: UDP
 - 3.4 Reliable Data Transfer, part 1 and part 2
 - 3.5 TCP, part 1 and part 2
 - 3.6 Principles of Congestion Control
 - 3.7 TCP Congestion Control
 - 3.8 Evolution of the Transport Layer