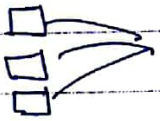
Chapter 3

Transport
→ port no
source destination
→ provide logical communication between processes in host

Network layer
→ Ip
Saure destiration
logical Communication between two hosts

layers is: 2 process → Tcp/Ip stack

multiplexing
collect data (جمع)



demultiplexing
العكس (تفرق)

socket: Ip address + port no

Demultiplexing

Connection demultiplexing
Tcp
use 4 tuple
- Source Ip
- source port no
- dest Ip
- dest port no

- receiver uses all 4 values to direct segment to appropriate socket
- each socket associated with a different connection client

connectionless demulti—
UDP
بيه destination and
use 2 tuple
- Dest Ip.
- Dest port no

IP/UDP datagrams with same dest port, but different Source ip address and/or source port numbers will be directed to same Socket.

UDP (simple)

- "no frills", "bare bones"
- "best effer service" segment may be → lost / delivered out of order
  - ⇒ No handshaking between UDP Sender, receiver
  - ⇒ each UDP segment handled independently of others.

Why using UDP ?

- no connection → ↓ RTT delay
- no connection state
- small header size → 8 byte
- no Congestion control

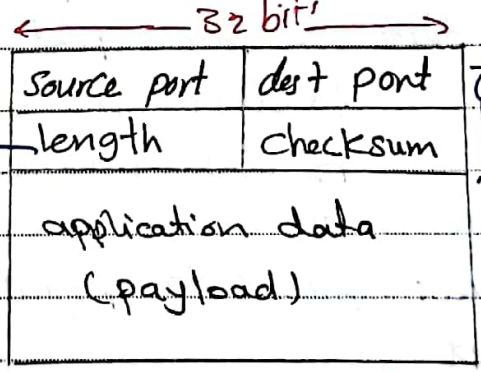
UDP use : - Streaming - DNS - Http/3 → QUIC ← UDP

- SNMP → Simple Network management protocol

Add reliability and congestion control at application layer if needed.

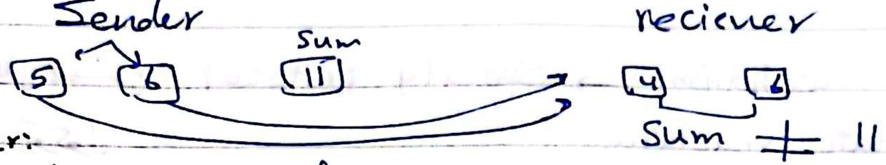
8 bytes = header

len header + mss



UDP segment format

Checksum → detection error (flipped bits).



Sender:
- treat contents of UDP as sequence of 16 bit
- ~~addtt~~

Checksum: addition (1's complement) of segment content

receiver: Compute checksum of received segment.
- Check if computed checksum equals checksum field value.

TCP → Reliable Data transfer
↓
transport layer
- Network → IP → UDP X reliable
- Data Link → Ethernet, PPP unreliable data transfer

الـ٢ تحت بيتعامل مع الاتنين طبقات دول يعتبر «unreliable»

↳ protocol → reliable data transfer (RDT)

→ Complexity of rdt ~~rel~~
↳ depend strongly → Characterstic of unreliable channel.

Sender
call data → rdt_send()
from above → pass to lower layer
unreliable ← udt_send() ⟶ ⟶

reciever
deliver_data()
↑
rdt_rcv()

rdt ازاى هبنى البروتوكول

« Building protocol » → name : rdt

FSM → Mechanism → Specify protocol الطريقه اللي هنشرح بيها البروتوكول

finite state machine

بيمشي → states

state ① —— event ——→ State 2

لازم يحصل action نتنقل لحاله تانية

{ Event Causing State transition
Action taken on State transition }

** Assume → rdt protocol في حالة unidirection Sender → reciever

Control flow → in both direction

يعني حكمه reciever يقول لـ sender إن فيه loss عادي

لكنه مش هيبعت داتا للـ sender.

rdt 1.0 first version

Assume → unreliable channel is perfect

X bit error

X loss

Sender rdt_send() event

[- wait for data from above
(App layer من)]

state ①

[transport layer → packet
packet لـ data هنقسم - make_pkt (data)
Udt_send ()]

يطلع state ① لـ

لازم يحصل action لـ

reciever

state ② [wait dat from below).

rdt_rcv (packet)

هيشيله اطرح → extract pkt (-,-)

deliver data ()

rdt 2.0 — Channel with bit errors

Channel may flip bits in packet ⇒ checksum

How recover error? ↵

(NACKs)
negative acknowledgements ← لو فيه مشكلة بعد ما طلعت Checksum] ARQ
pkt received ok (ACKs) ← لو مافيش

Automatic Repeat Request ↵

protocol:
- error detection
- Reciever feedback ← تعليقات جابات
- retransmission

Stop and wait → Sender send one pkt and wait for respond from reciever.

FSM

⇒ Sender

State ①: wait data from above
State ②: wait for ACK, NACK

rdt_send (data) → event
packet = make_pkt (data) } action
udt_send (packet)

next pkt لـ هروح لـ ثانية بناءً على الرد

(if) → rdt_rcv(packet) && is ACK → هرجع للمستنى إلى State ①

(if) → rdt_rcv(packet) && is NACK → udt_send (packet)
هبعت نفس الباكت

⇒ reciever

State ①: wait data from below

(if) rdt_rcv(pkt) && is not corrupt ( )
extract pkt (packet, data), deliver (data)
udt_send (ACK)

(if) rdt_rcv(pkt) && is corrupt( )
udt_send (NAck)