

Advanced Network Technologies

Application layer

Transport layer

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Dr. Wei Bao | Lecturer
School of Computer Science



THE UNIVERSITY OF
SYDNEY



Assignment Project Exam Help

Peer-to-Peer

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Pure peer-to-peer model architecture

- › no always-on server
- › arbitrary end systems directly communicate
- › peers are intermittently connected and change addresses

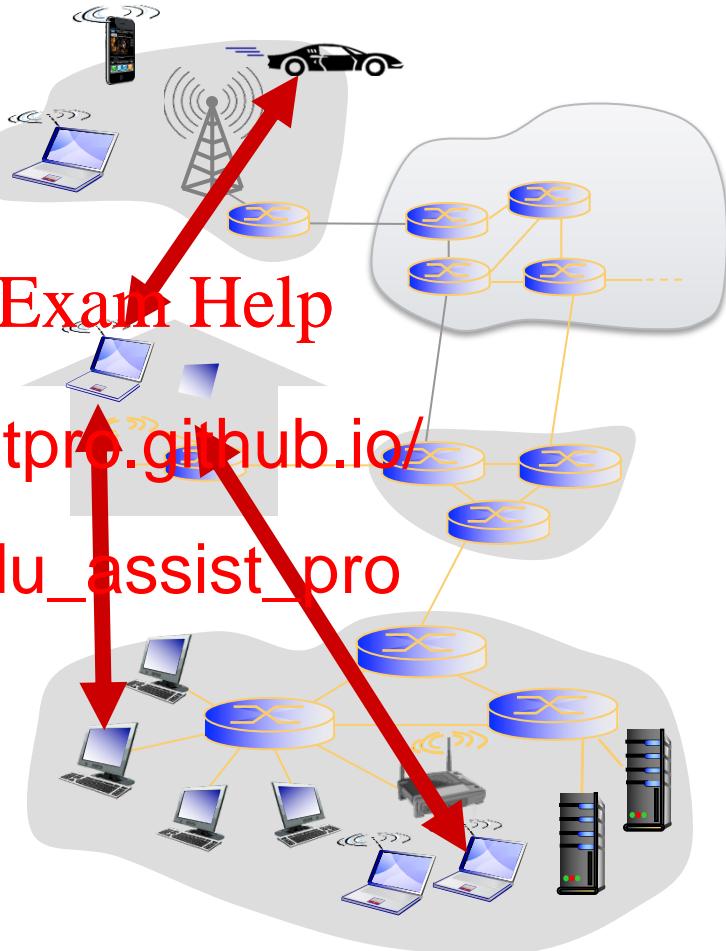
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

examples:

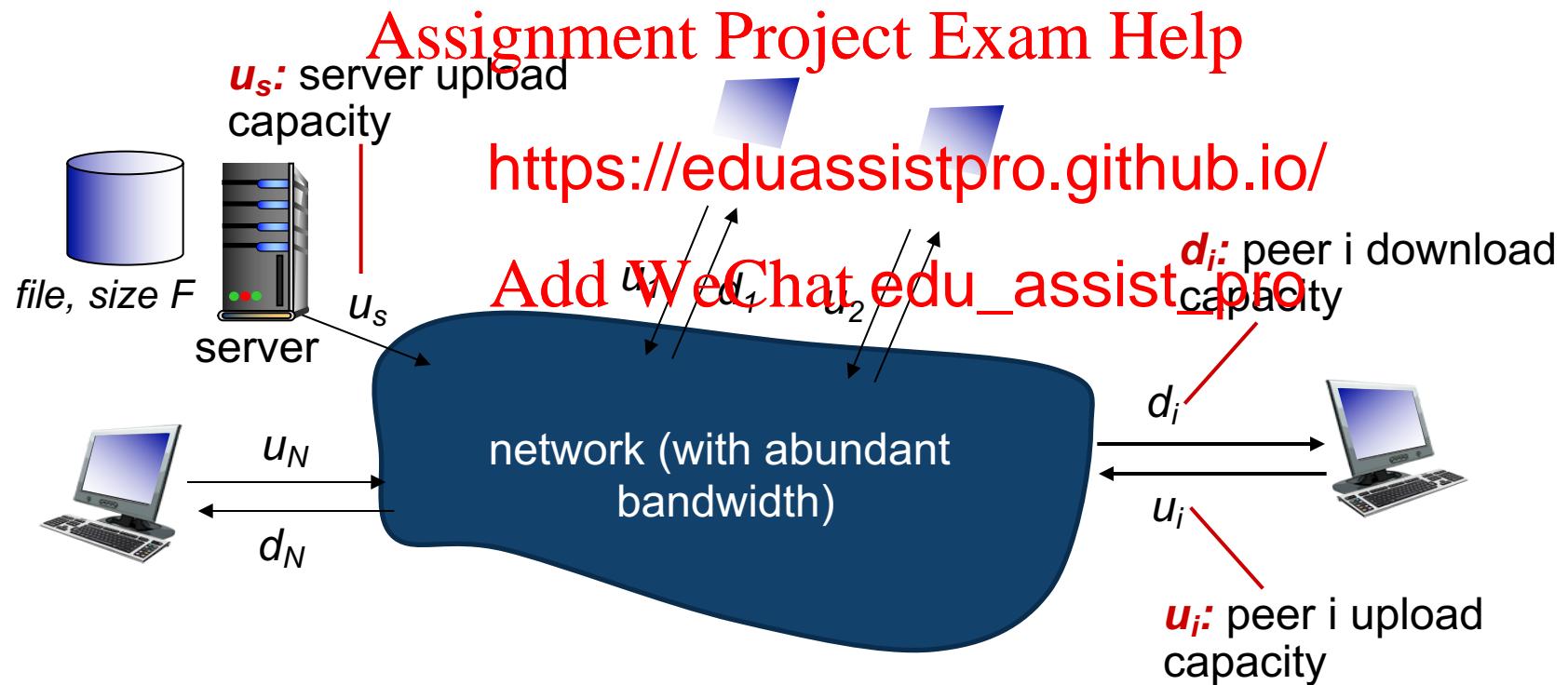
- file distribution (BitTorrent)
- Streaming (Zattoo, KanKan)
- VoIP (Skype)



File distribution: client-server vs. p2p

Question: how much time to distribute file (size F) from one server to N peers?

- peer upload/download capacity is limited resource



File distribution time: client-server

› **server transmission:** must sequentially send (upload) N file copies:

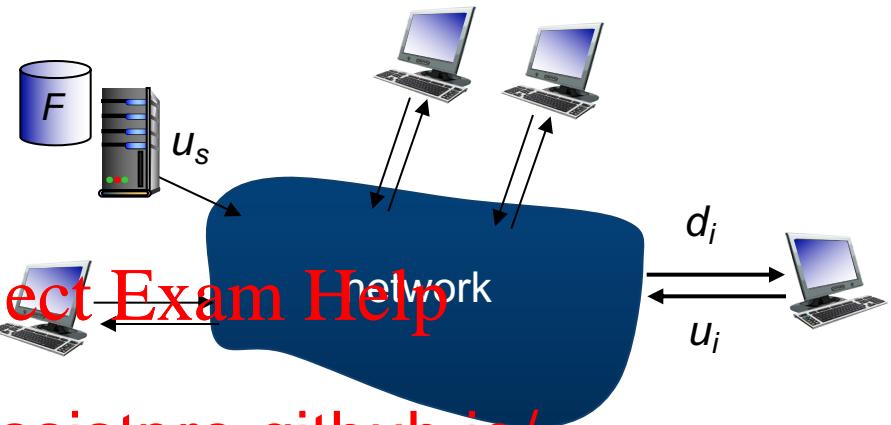
- time to send one copy: F/u_s
- time to send N copies

❖ **client:** each client download file co

- d_{min} = min client download rate
- (worst case) client download time: F/d_{min}

Assignment Project Exam Help

Add WeChat edu_assist_pro



*time to distribute F
to N clients using
client-server approach*

$$D_{c-s} \geq \max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

› **server transmission:** must upload at least one copy

- time to send one copy: F/u_s

- ❖ **client:** each client must download file copy

- client download time

- ❖ **clients:** as aggregate

- Max upload rate $u_s + \sum u_i$

- $NF/(u_s + \sum u_i)$

Assignment Project Exam Help
<https://eduassistpro.github.io/> = upload NF bits

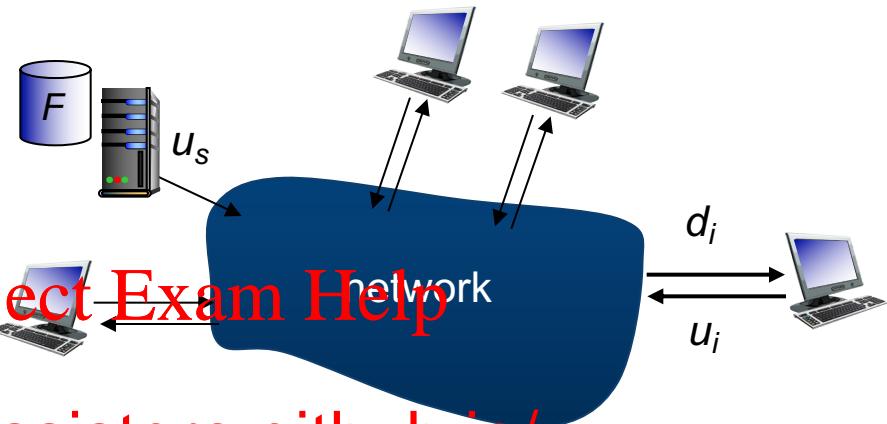
Add WeChat **edu_assist_pro**

time to distribute F to N clients using P2P approach

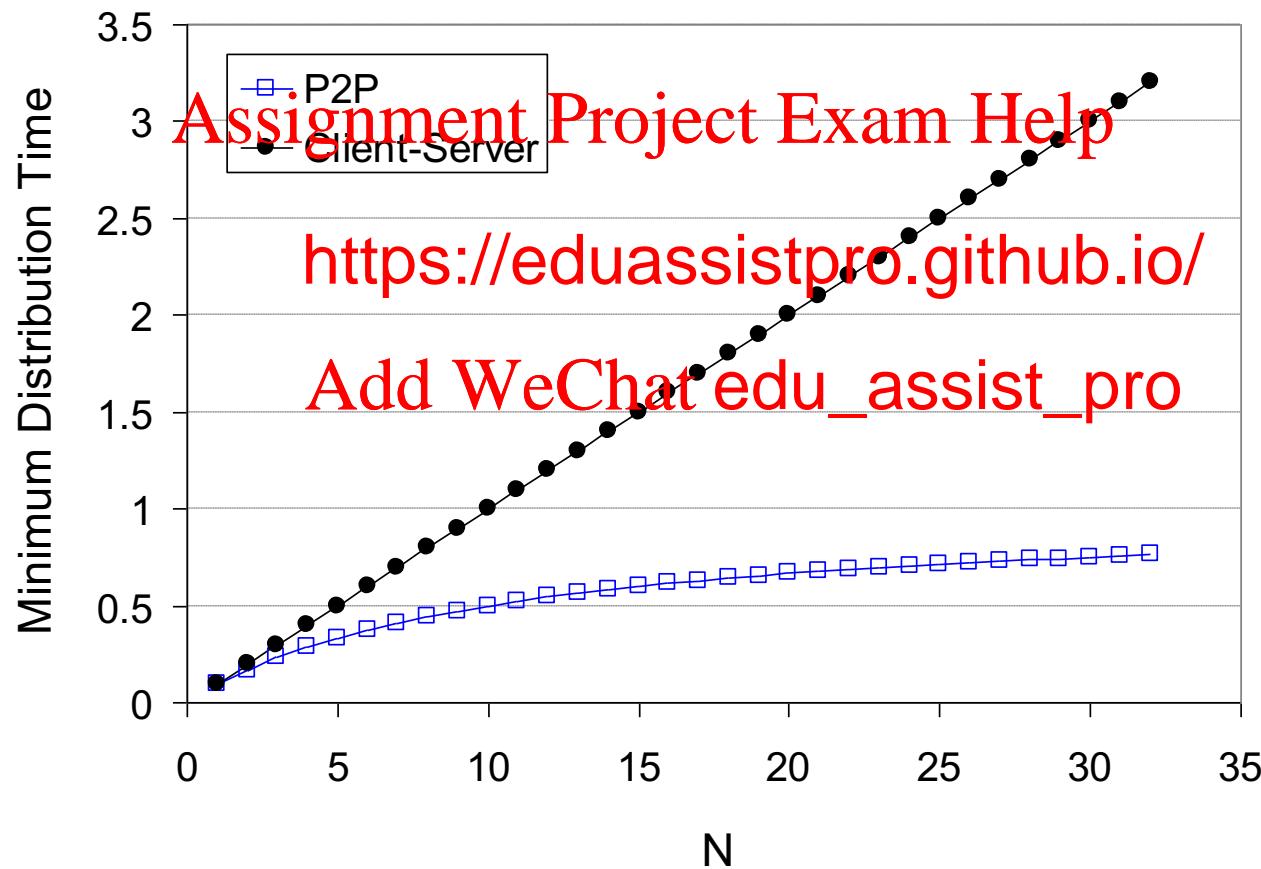
$$D_{P2P} \geq \max\{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$$

increases linearly in N ...

... but so does this, as each peer brings service capacity



client upload rate = u , $F/u = 1$ hour, $u_s = 10u$, $d_{min} \geq u_s$



BitTorrent, a file sharing application

- › 20% of European internet traffic in 2012.
- › Used for Linux distribution, software patches, distributing movies
- › Goal: quickly replicate large files to large number of clients



<https://eduassistpro.github.io/>

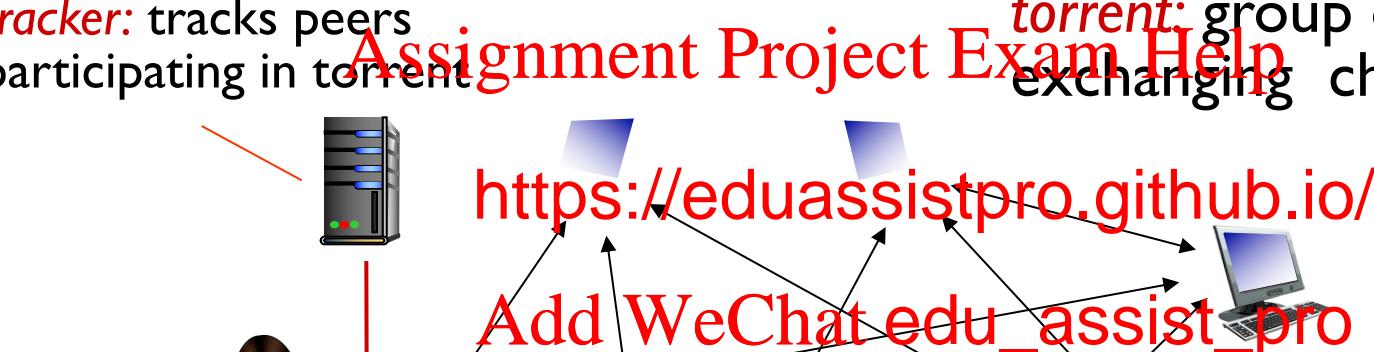
- › Web server hosts a .torrent file (w/ file I tracker's URL...)
- › A tracker tracks downloaders/owners o **Add WeChat edu_assist_pro**
- › Files are divided into chunks (256kB-1MB)
- › Downloaders download chunks from themselves (and owners)
- › Tit-for-tat: the more one shares (**server**), the faster it can download (**client**)

- › file divided into 256KB chunks
- › peers in torrent send/receive file chunks



tracker: tracks peers
participating in torrent

torrent: group of peers
exchanging chunks of a file



› peer joining torrent:

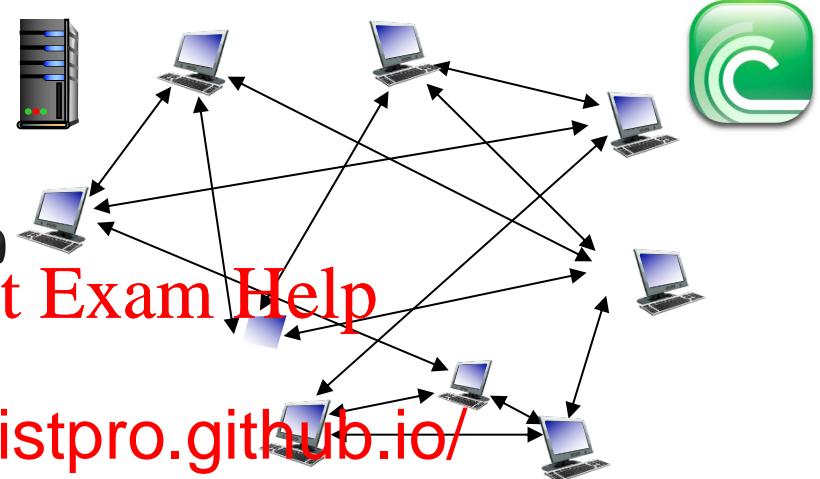
- has no chunks, but will accumulate them over time from other peers

Assignment Project Exam Help

- registers with trac

peers, connects to <https://eduassistpro.github.io/>
("neighbors")

Add WeChat edu_assist_pro



› while downloading, peer uploads chunks to other peers

› peer may change peers with whom it exchanges chunks

› *churn*: peers may come and go

› once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

requesting chunks:

- › at any given time, different peers have different subsets of file chunks
- › periodically, Alice peer for list of chunks they have
- › Alice requests missing chunks from peers, rarest first

sending chunks: tit-for-tat

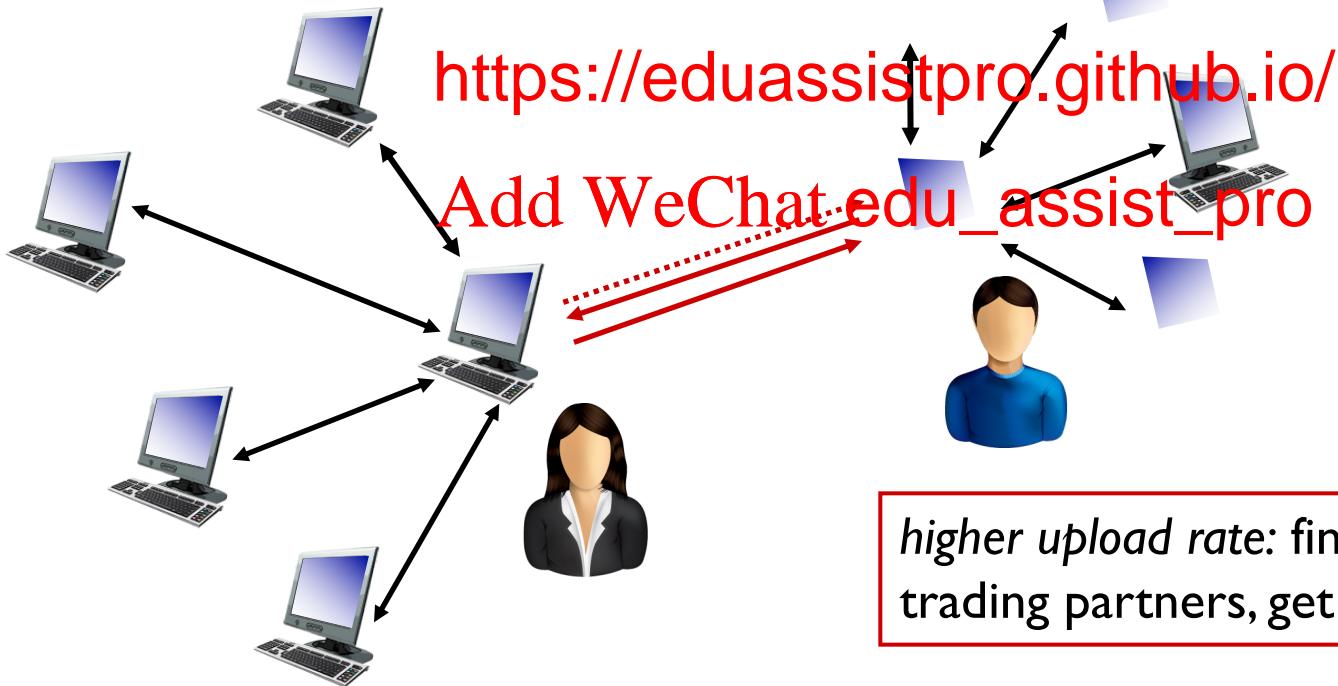
- › Alice sends chunks to those four peers currently sending her chunks *at highest rate*
- https://eduassistpro.github.io/
the choked by Alice (do not
nks from her)
- Add WeChat **edu_assist_pro**
4 every 10 secs
- › every 30 secs: randomly select another peer, starts sending chunks
 - › “optimistically unchoke” this peer
 - › newly chosen peer may join top 4



- (1) Alice “optimistically unchoke” Bob
- (2) Alice becomes one of Bob’s top-four providers; Bob reciprocates
- (3) Bob becomes one of Alice’s top-four providers



Assignment Project Exam Help





Distributed Hash Table

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

- › DHT: a *distributed P2P database*
- › database has **(key, value)** pairs; examples:
 - key: social **Assignment Project Exam Help** human name
- › distribute th **<https://eduassistpro.github.io/>** over the many peers
 - Add WeChat **edu_assist_pro**
- › a peer **queries** DHT with key
 - DHT returns values that match the key
- › peers can also **insert** **(key, value)** pairs



- › Assign the keys
- › Lookup the keys

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

› central issue:

- assigning (key, value) pairs to peers.

› basic idea: Assignment Project Exam Help

- Key: generate <https://eduassistpro.github.io/>
- Assign an integer ID to a WeChat account `edu_assist_pro`
- put (key,value) pair in the peer that is closest to the key

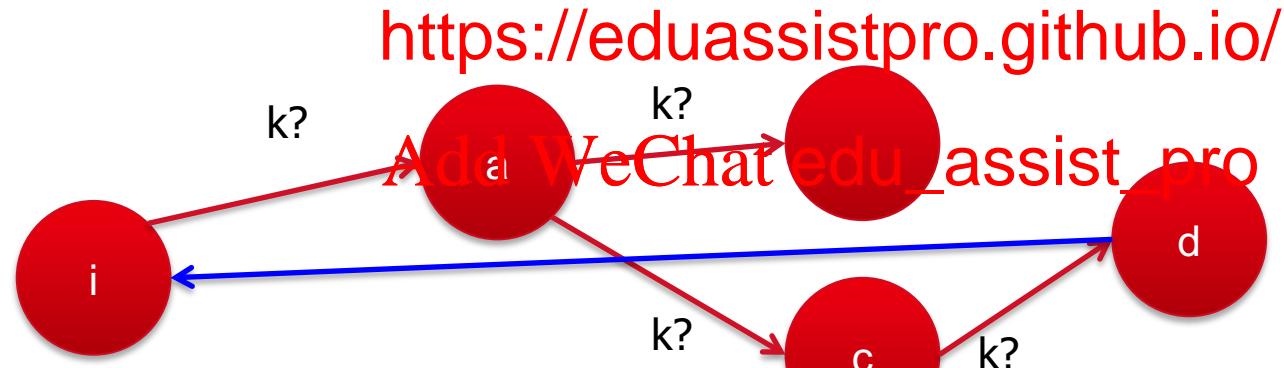
- › **distance**: assign integer identifier to each peer in range $[0, 2^n - 1]$ for some n .
 - each identifier represented by n bits
- › Each key to be <https://eduassistpro.github.io/> assigned to peer in range $[0, 2^n - 1]$
- › to get integer
 - A hash function is any function that takes an input of arbitrary size and produces a fixed-size output (e.g., an integer in $[0, 2^n - 1]$).
 - e.g., $15 = \text{hash}(\text{"Led Zeppelin IV"})$
 - this is why it is referred to as a *distributed “hash” table*

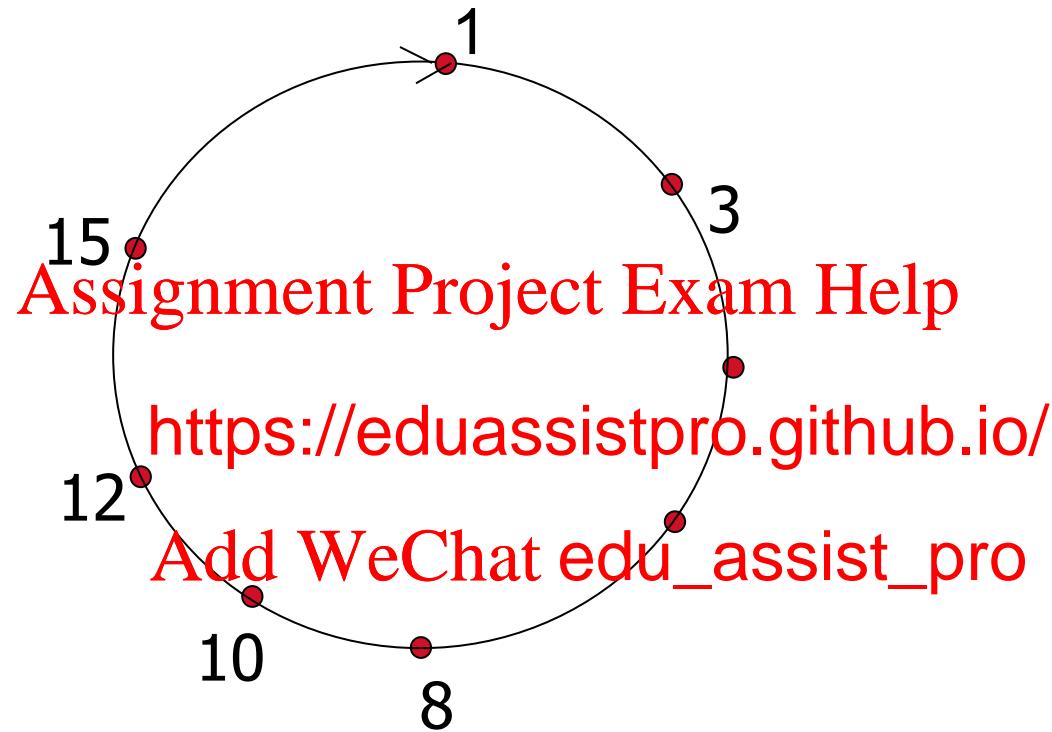
- › rule: assign key to the peer that has the **closest** ID.
- › Here: closest ~~Assign the Project Leader or~~ of the key.
<https://eduassistpro.github.io/>
- › e.g., $n = 4$; peers: 1, 3, 4, 12, 14;
~~Add WeChat edu_assist_pro~~
 - key = 13, then successor peer = 14
 - key = 15, then successor peer = 1

Goal: to provide a distributed lookup service returning the host that owns the key

- Given a key, find the host that owns the key

Assignment Project Exam Help



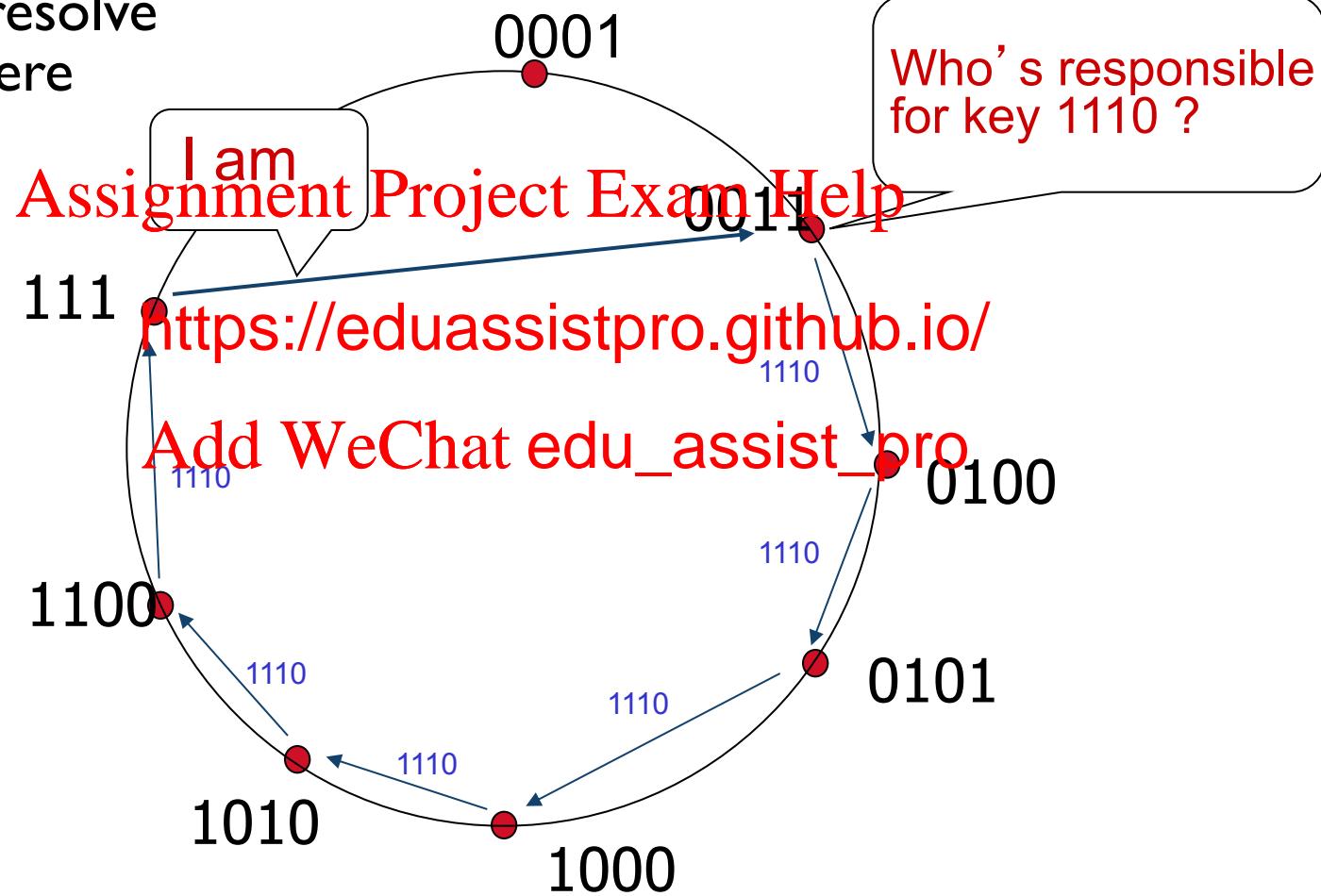


- › each peer *only* aware of immediate successor.

$O(N)$ messages
on average to resolve
query, when there
are N peers

key 1110 is
stored at node
1111

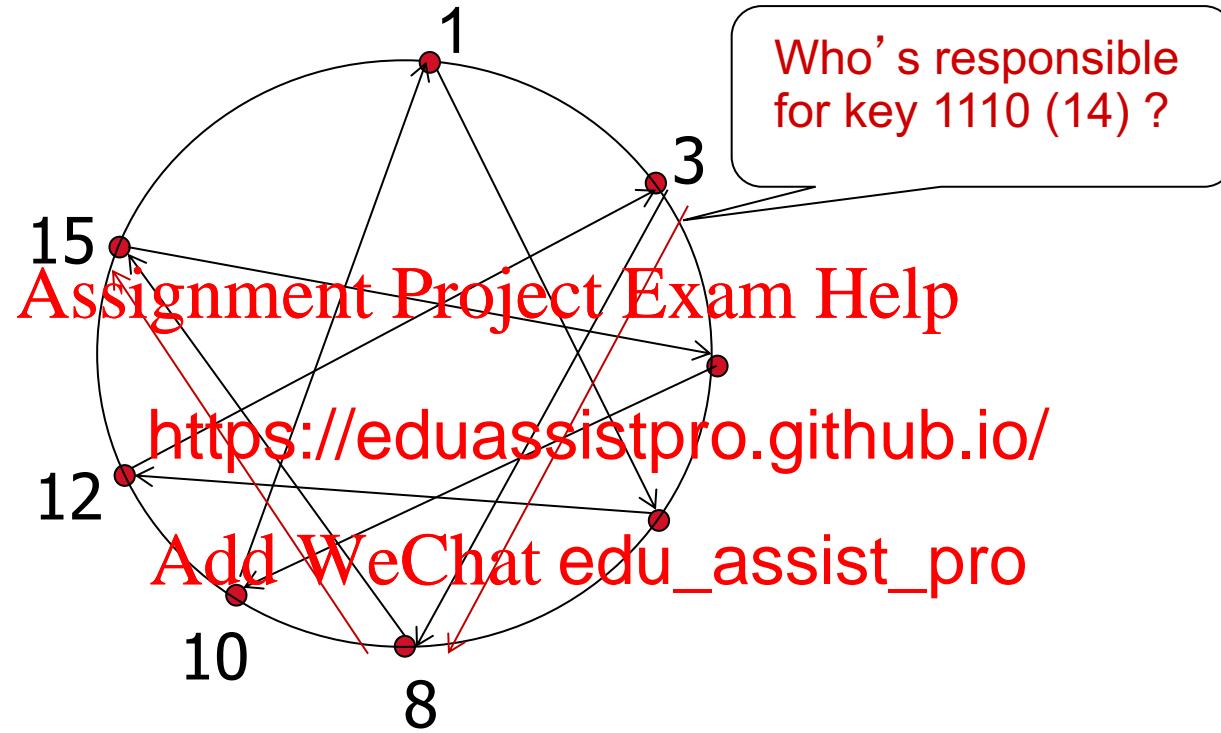
Define closest
as closest
successor



Example: Chord is an example of a Distributed Hash Table (DHT)

As a node:

- › I have a successor peer **Assignment Project Exam Help**
- › I have a predecessor p
- › I have some shortcuts **<https://eduassistpro.github.io/>** to speedup delivery of requests **Add WeChat edu_assist_pro**
- › Chord: A scalable peer-to-peer lookup service for internet applications. Stoica et al. SIGCOMM 2001.



- › each peer keeps track of predecessor, successor, short cuts.
- › reduced from 6 to 2 messages.
- › possible to design shortcuts so $O(\log N)$ neighbors, $O(\log N)$ messages in query



Assignment Project Exam Help **Transport Layer**

<https://eduassistpro.github.io/>

Add WeChat **edu_assist_pro**

our goals:

- › understand principles behind transport layer services:
 - multiplexing, demultiplexing
 - reliable data transfer
 - flow control
 - congestion control
- › learn about Internet protocols:
 - Assignment Project Exam Help
 - <https://eduassistpro.github.io/>
 - Add WeChat
 - TCP connectionless transport
 - TCP connection-oriented port
 - TCP congestion control

- › Transport-layer services

Assignment Project Exam Help

- › Multiplexing/demultiplexing

<https://eduassistpro.github.io/>

- › Connectionless transport (UDP)

Add WeChat edu_assist_pro

- › Principles of reliable data transfer

- › TCP protocol



Assignment Project Exam Help Transport Services

<https://eduassistpro.github.io/>

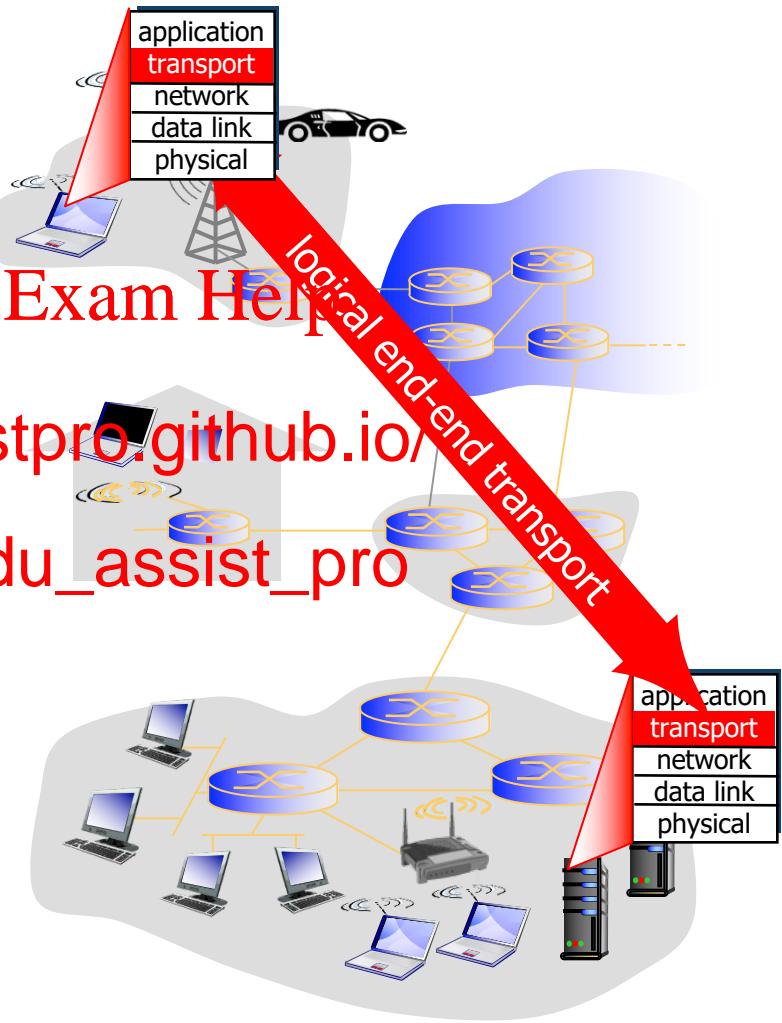
Add WeChat edu_assist_pro

- › provide *logical communication* between app processes running on different hosts
- › transport protocols run in end systems
 - send side: breaks app into *segments*, passes to network layer
 - rcv side: reassembles segments into messages, passes to app layer
- › more than one transport protocol available to apps
 - Internet: TCP and UDP

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro



› **network layer:** host-to-host communication

- best-effort ~~Assignment Project Exam Help~~

<https://eduassistpro.github.io/>

› **transport layer:** ~~Add WeChat~~ process-to-process communication

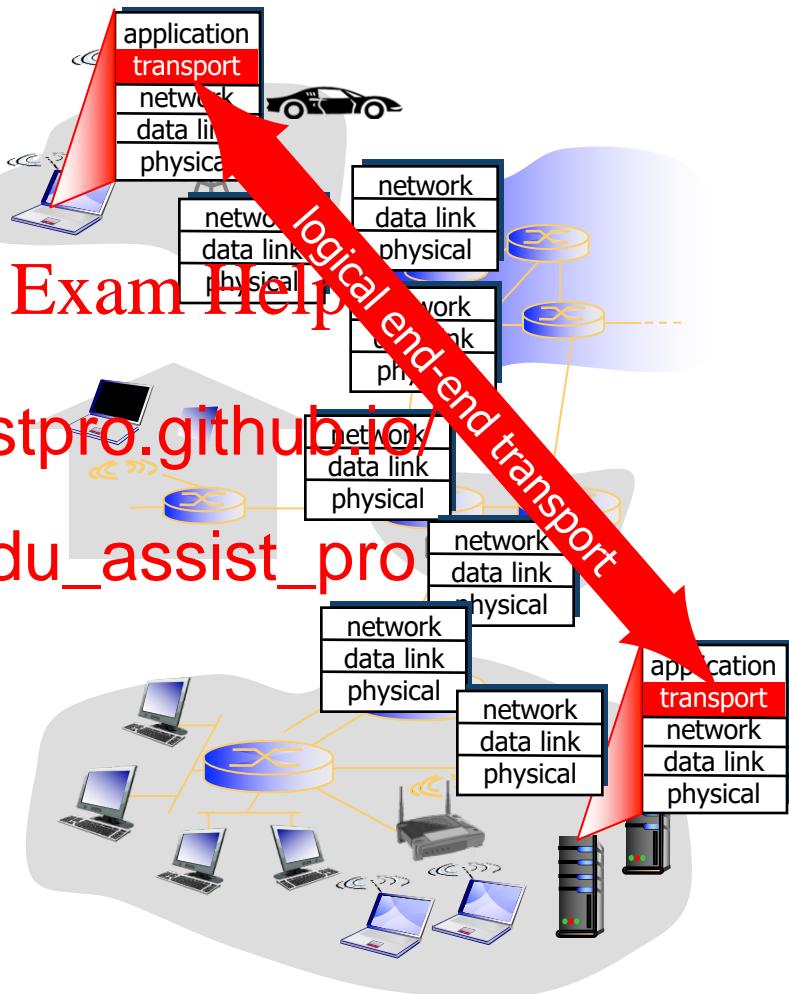
- relies on, enhances, network layer services

- › IP: best effort service
- › reliable, in-order delivery (TCP)
 - congestion control
 - flow control
 - connection setup
- › unreliable, unordered delivery: UDP
 - no-frills extension of “best-effort” IP
- › services not available:
 - delay guarantees
 - bandwidth guarantees

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro





Assignment Project Exam Help Transport Services

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

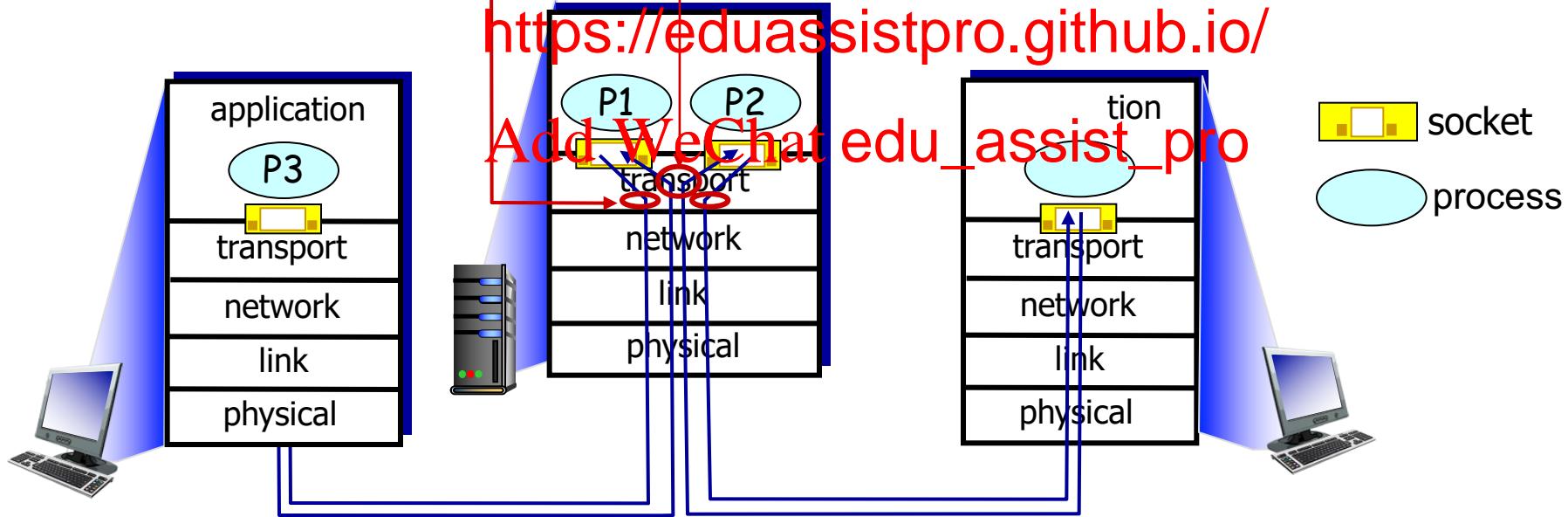
multiplexing at sender:

handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:

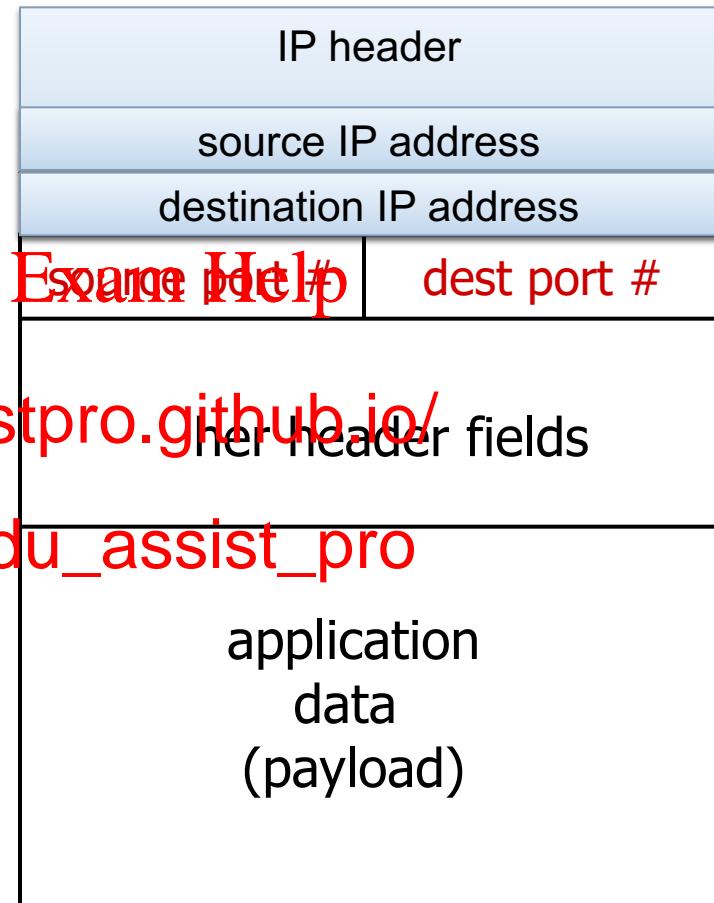
use header info to deliver received segments to correct socket

Assignment Project Exam Help



How demultiplexing works

- › host receives IP datagrams
 - each datagram has source IP address, destination IP address
 - each datagram carries one transport-layer segment
 - each segment has source port number



TCP/UDP segment format

› Receiver

- › *recall:* created socket has host-local port #:

› Sender

- ❖ *recall:* when creating datagram to send into UDP socket, must specify
 - destination IP address

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat `edu_assist_pro`

- › when host receives UDP segment:

- Checks destination port # in segment
- directs UDP segment to socket with that port #



with same dest. IP address, dest. port #, but different source IP addresses and/or source port numbers will be directed to **same socket** at dest

Connectionless demux: example

```
DatagramSocket serverSocket =
    new DatagramSocket(6428);

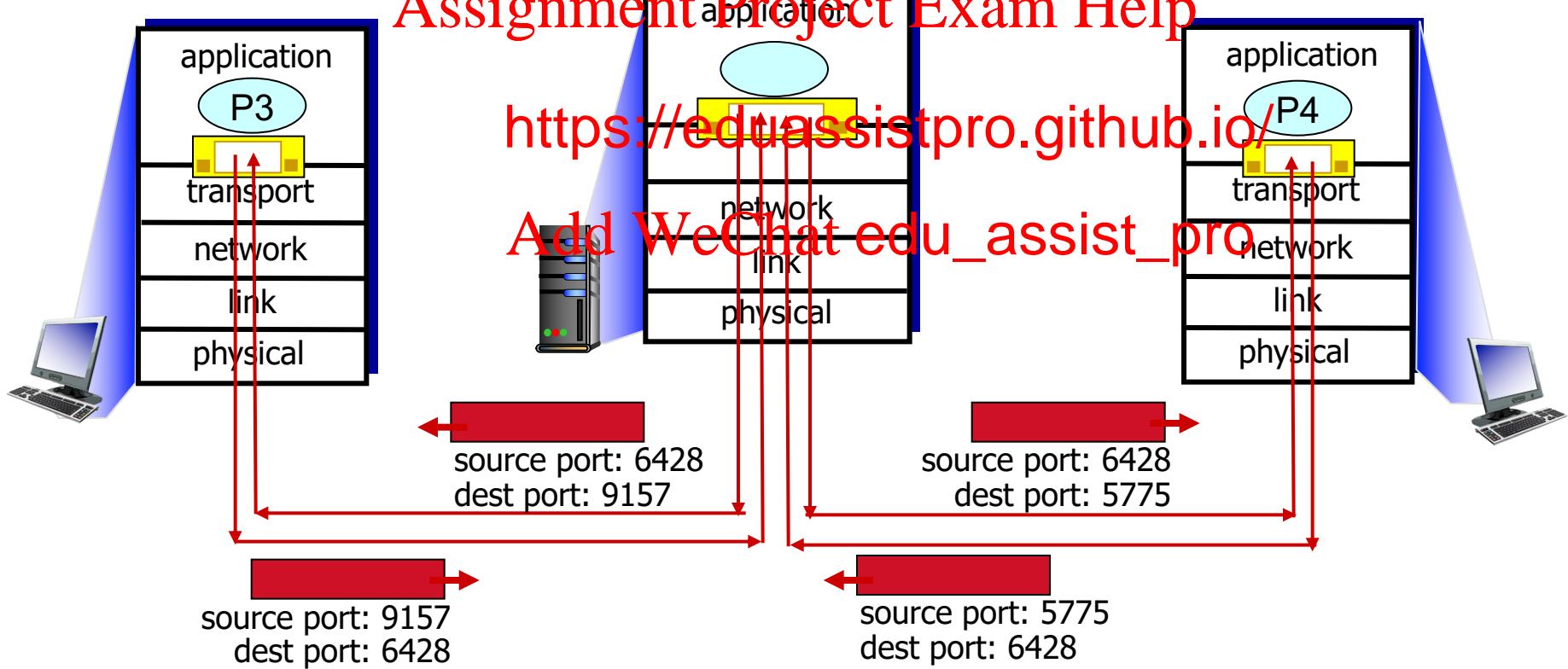
DatagramSocket mySocket2 =
new DatagramSocket(9157);
```

```
DatagramSocket mySocket1 =
new DatagramSocket(5775);
```

Assignment Project Exam Help

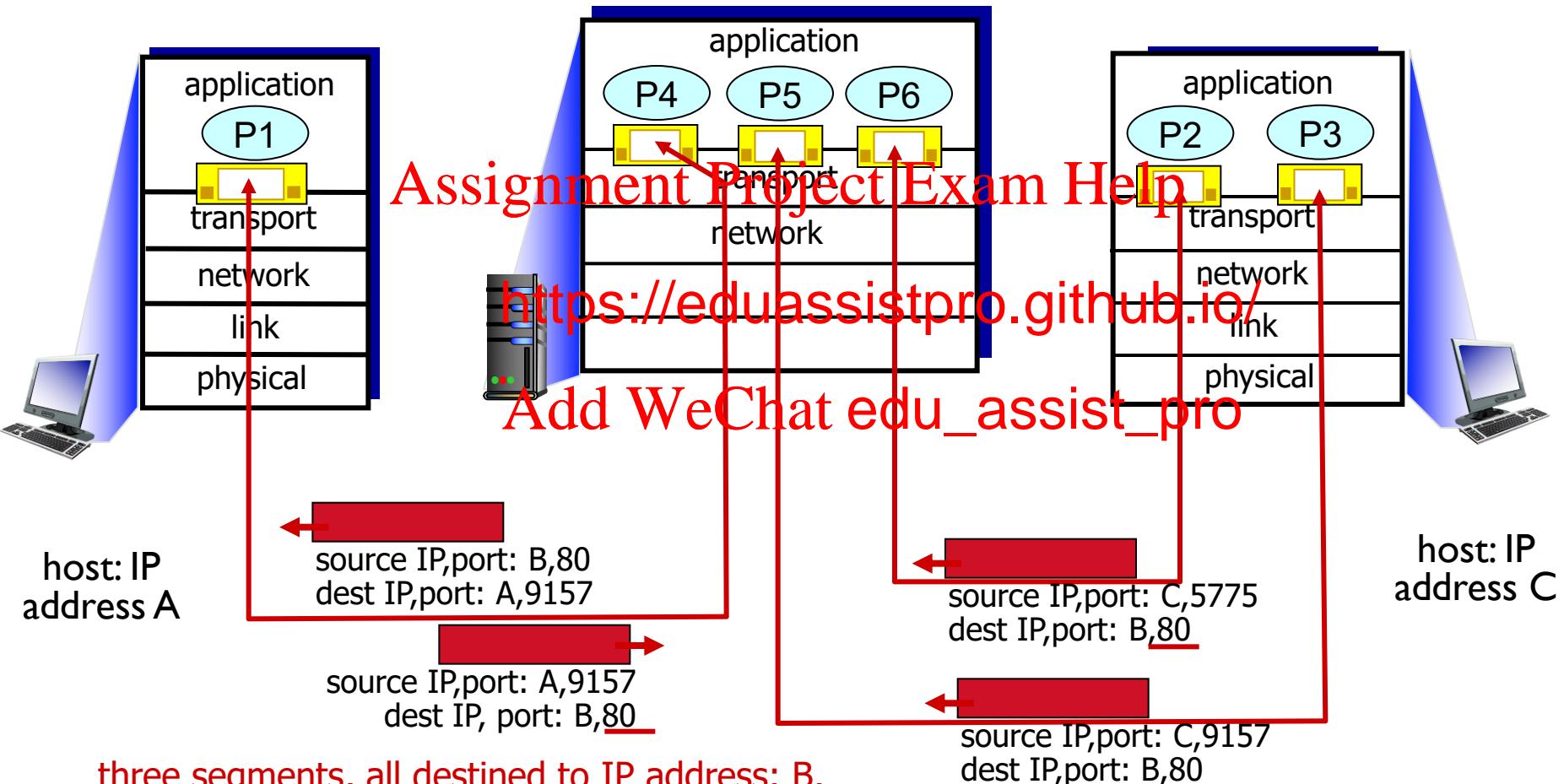
<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro



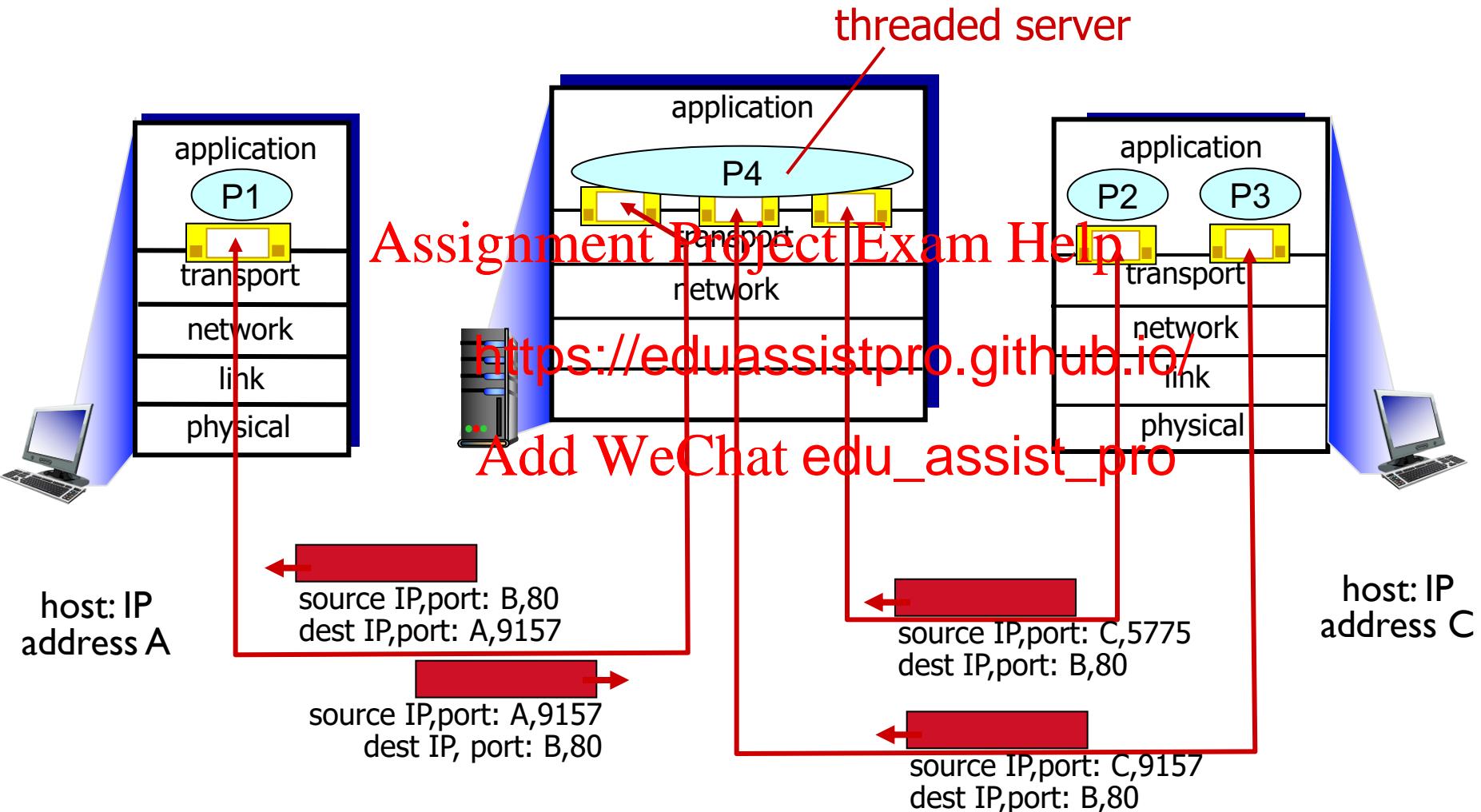
- › TCP socket identified by 4-tuple:
 - source IP address
 - source port number
 - dest IP address
 - dest port number
- › demux: receiver uses all four values to direct segment to appropriate socket
- › server host may support many simultaneous TCP sockets:
 - Assignment Project Exam Help
 - et identified by its socket
 - <https://eduassistpro.github.io/>
 - Add WeChat `edu_assist_pro` to have different clients for each connecting client
 - non-persistent HTTP will have different socket for each request

Connection-oriented demux: example



three segments, all destined to IP address: B,
 dest port: 80 are demultiplexed to *different* sockets

Connection-oriented demux: example





Connectionless Transport UDP

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

UDP: User Datagram Protocol [RFC 768]

- › “no frills,” Internet transport protocol
- › “best effort” service, UDP segments may be
 - lost
 - delivered out-of-order
- › **connectionless:**
 - no handshaking between UDP sender, receiver
 - each UDP segment handled independently of others

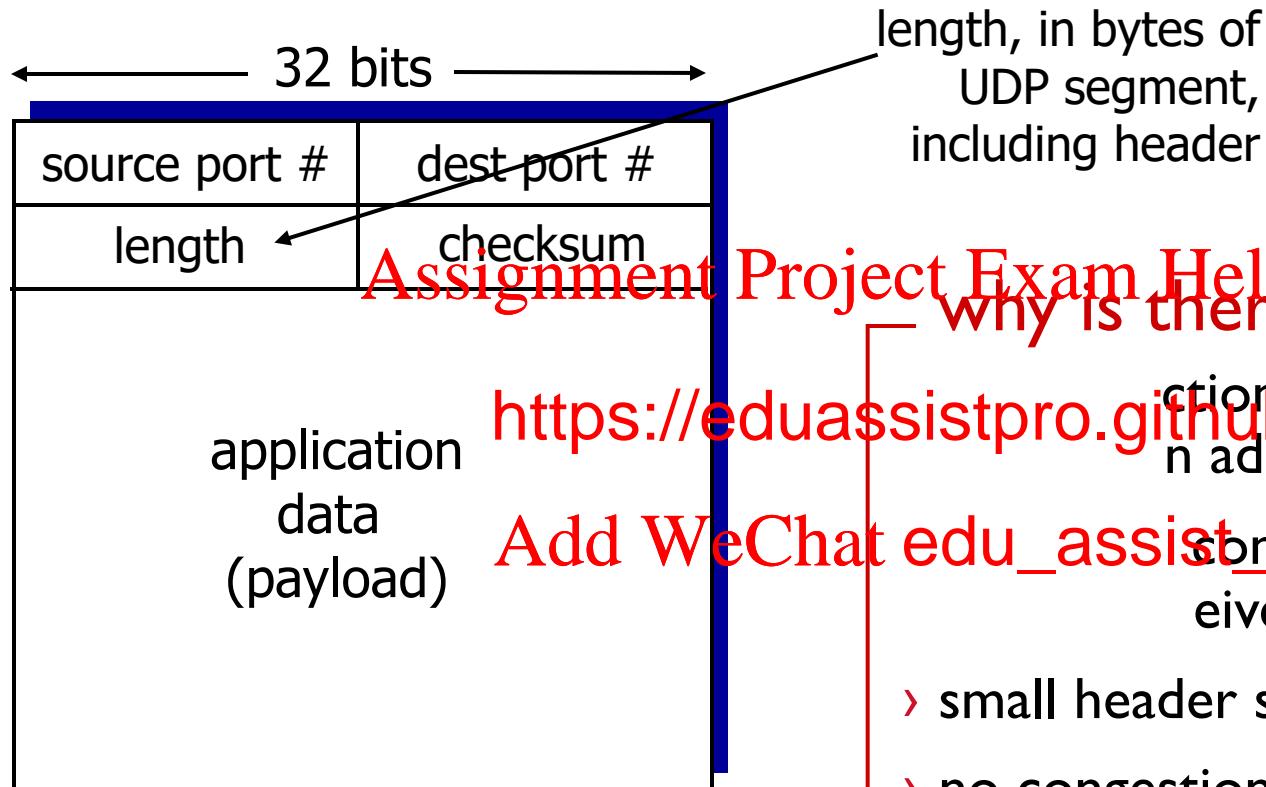
❖ UDP use:

- streaming multimedia apps (loss tolerant, rate sensitive)
- DNS

<https://eduassistpro.github.io/>

❖ Add WeChat edu_assist_pro

- application-specific error recovery!



UDP segment format

length, in bytes of
UDP segment,
including header

Assignment Project Exam Help

why is there a UDP?

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

ction establishment
(n add delay)

connection state at
eiver

- › small header size
- › no congestion control: UDP can blast away as fast as desired

Goal: detect “errors” (e.g., flipped bits) in transmitted segment

sender: Assignment Project Exam Help

- › treat segment contents including header fields as sequence of 16-bit words
 - › sum: addition (one's complement sum) of segment contents
 - › checksum: complement of sum
 - › sender puts checksum value into UDP checksum field
- receiver: calculate checksum of received words
- <https://eduassistpro.github.io/>
- Add WeChat `edu_assist_pro` for support
- computed checksum value:
- NO - error detected
 - YES - no error detected.

example: add two 16-bit integers

1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
1AssignmentProjectExamHelp

carryout 1 0 1 https://eduassistpro.github.io/

wraparound 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 →

Add WeChat edu_assist_pro

sum 1 0 1 1 1 0 1 1 1 0 0

checksum 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result



Principles of Reliable Data Transfer

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro



- › important in application, transport, link layers
 - top-10 list of important networking topics!

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

- › characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)



- › important in application, transport, link layers
 - top-10 list of important networking topics!

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

- › characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

- › important in application, transport, link layers
 - top-10 list of important networking topics!

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat `edu_assist_pro`

Network
layer

- › characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

`rdt_send()` : called from above,
(e.g., by app.). Passed data to
deliver to receiver upper layer

`deliver_data()` : called by
rdt to deliver data to upper

Assignment Project Exam Help

send
side

<https://eduassistpro.github.io/> receive
side

Add WeChat `edu_assist_pro`

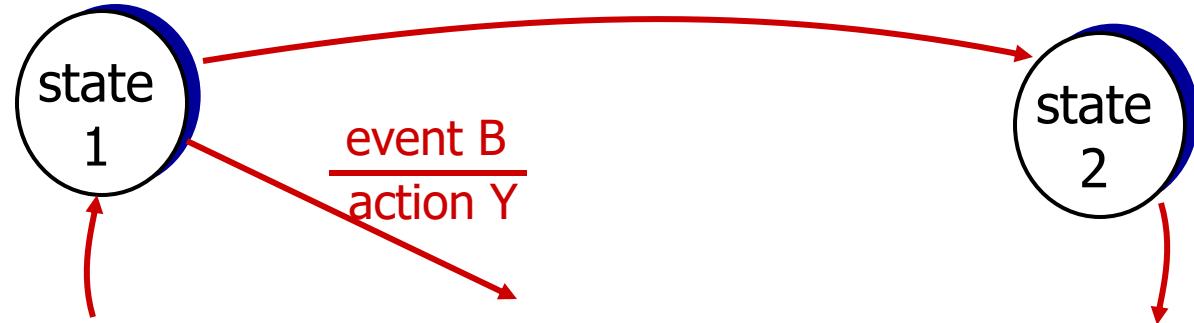
`udt_send()` : called by rdt,
to transfer packet over
unreliable channel to receiver

`rdt_rcv()` : called when packet
arrives on rcv-side of channel

We will:

- › incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- › consider only unidirectional data transfer
 - but control info <https://eduassistpro.github.io/>
- › use finite state machines (FS) toify sender, receiver

state: when in this “state”, next state and action uniquely determined by next event



› underlying channel perfectly reliable

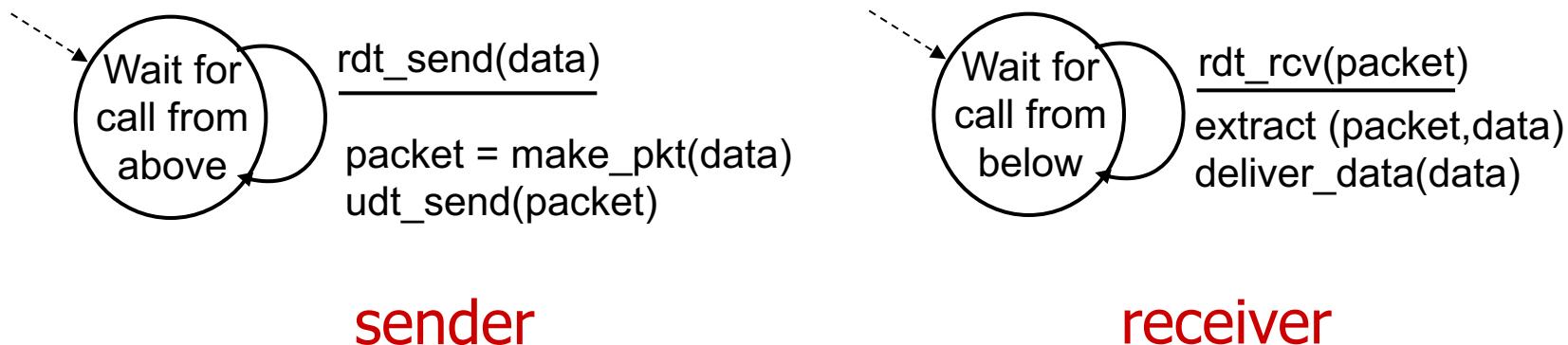
- no bit errors
- no loss of packets

Assignment Project Exam Help

› separate FSMs for sender and receiver

- sender sends data to receiver
- receiver reads data from sender

Add WeChat edu_assist_pro

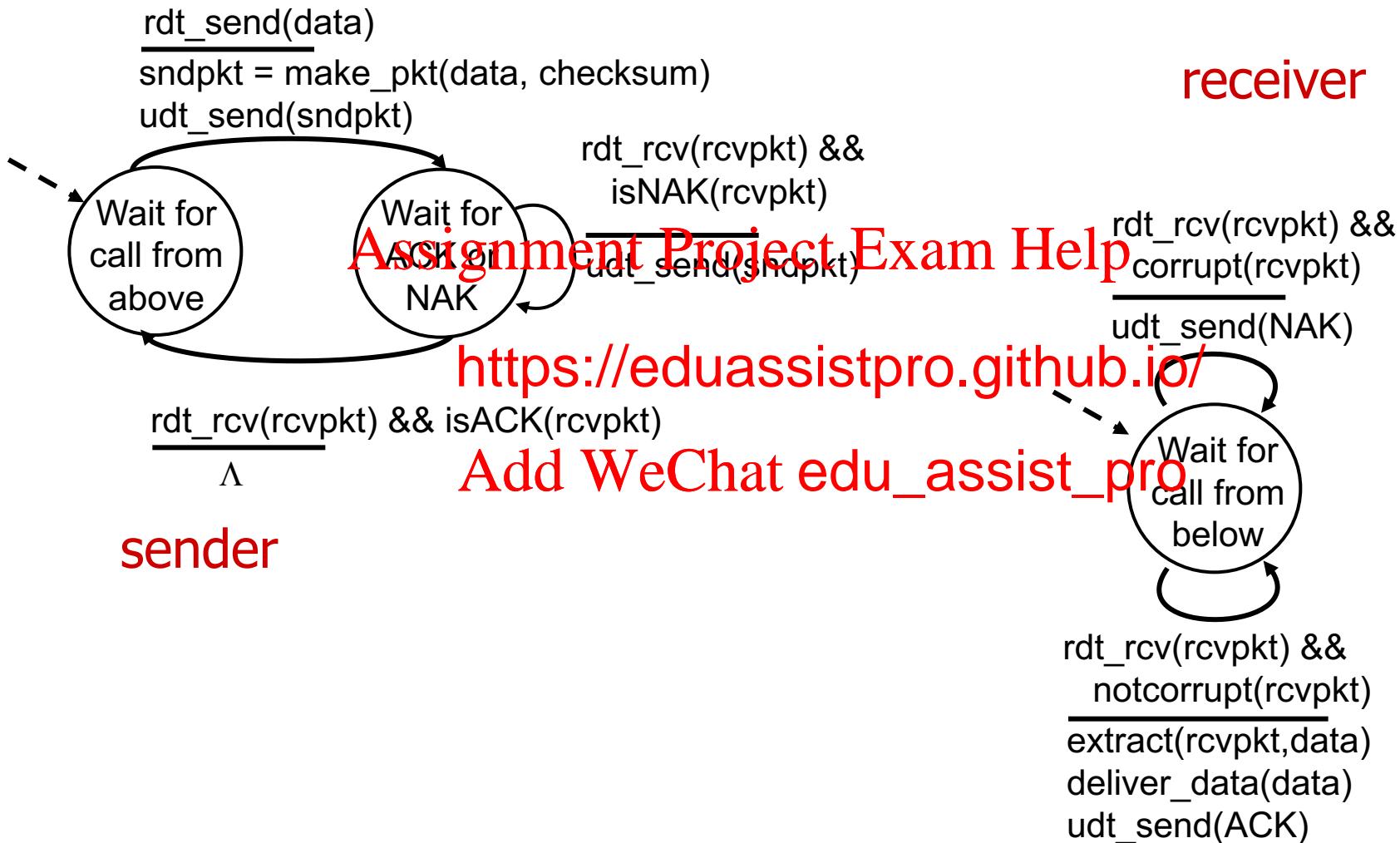


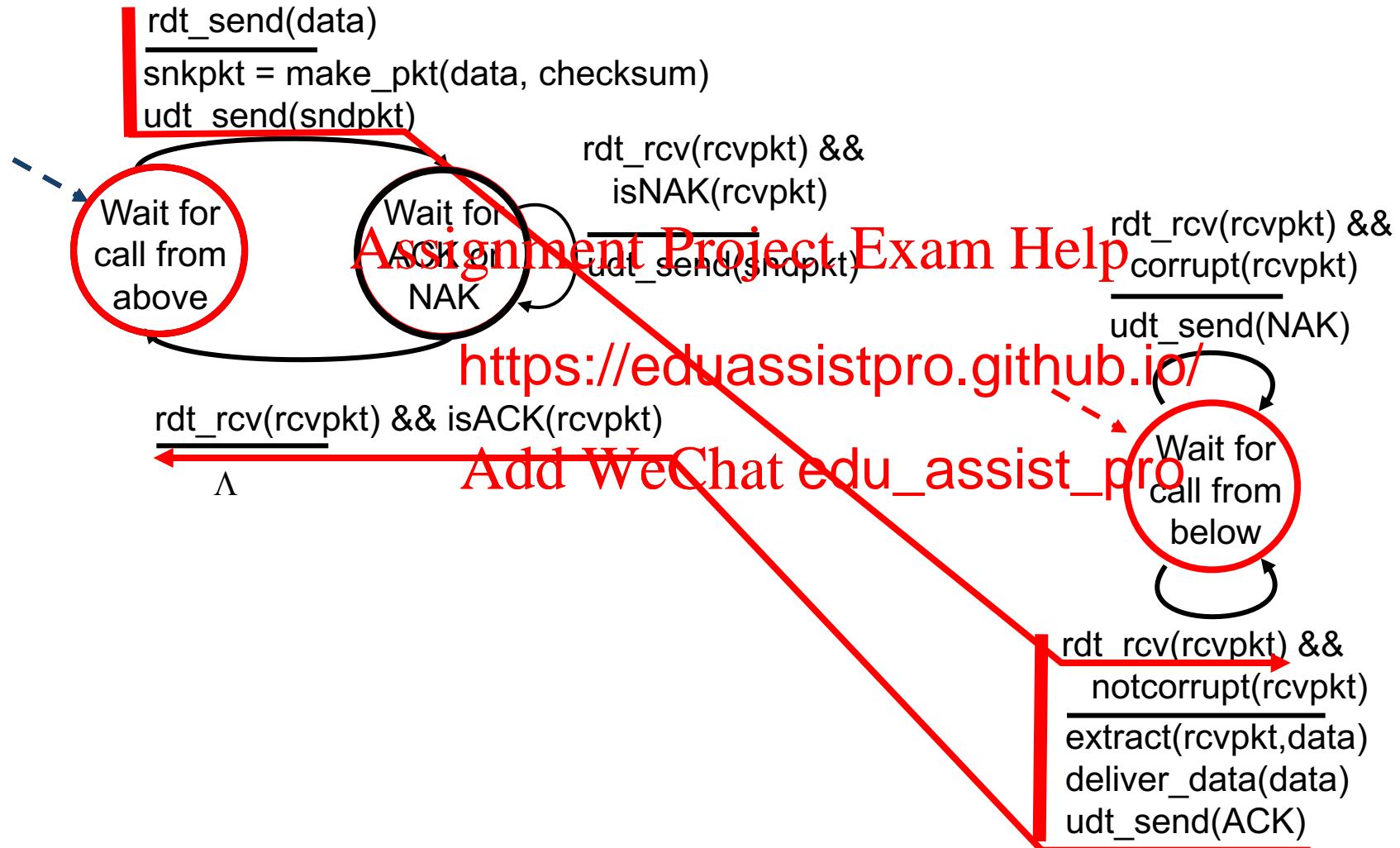
- › underlying channel may flip bits in packet
 - checksum to detect bit errors
- › the question: how to recover from errors:
Assignment Project Exam Help

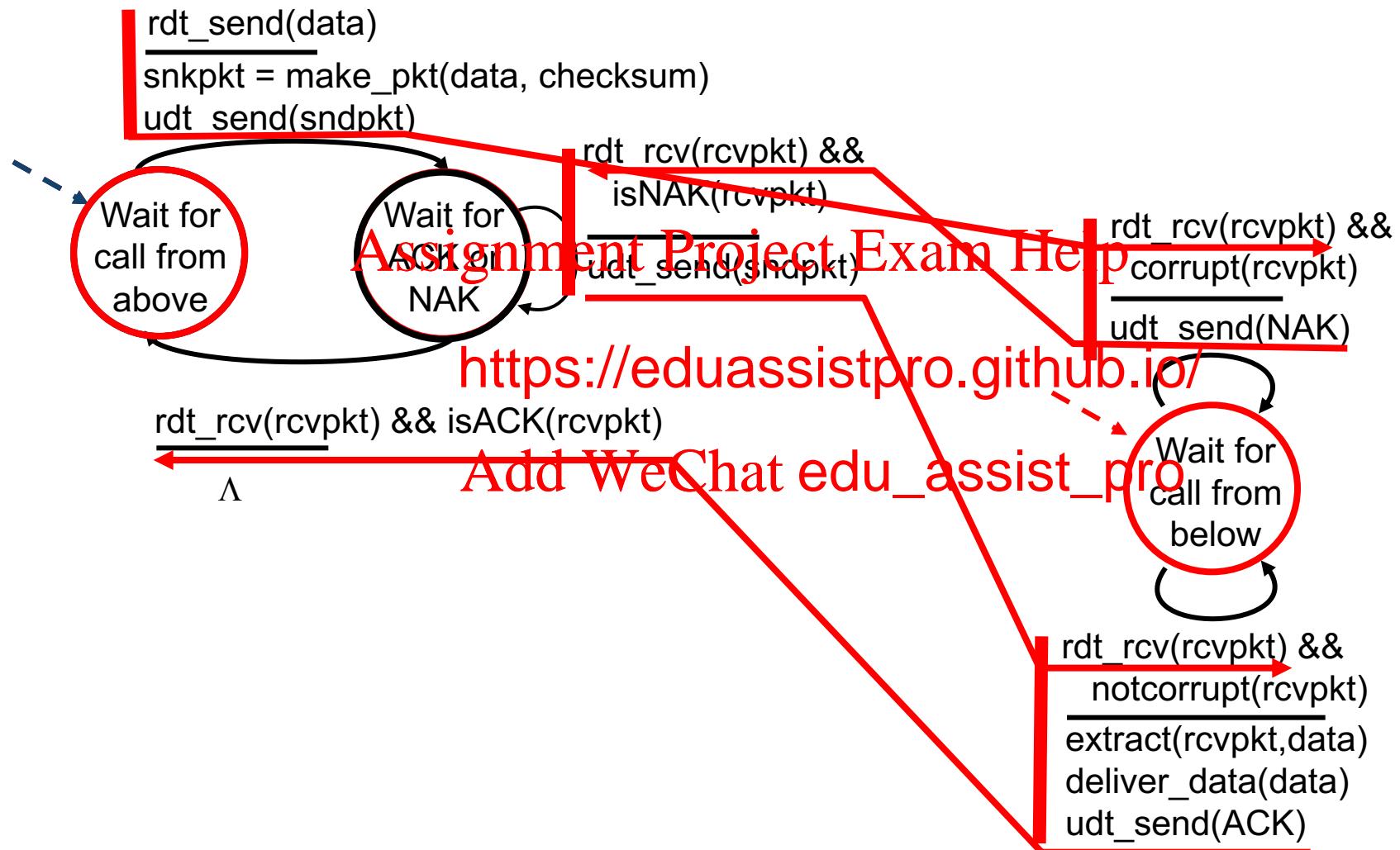
<https://eduassistpro.github.io/>

*How do humans Add WeChat edu_assist “errors”
during conversation?*

- › underlying channel may flip bits in packet
 - checksum to detect bit errors
- › the question: how to recover from errors:
 - acknowledgements (ACKs): receiver explicitly tells sender that pkt received <https://eduassistpro.github.io/>
 - negative acknowledgements (NAKs): sender tells receiver that pkt had errors [Add WeChat edu_assist_pro](#)
 - sender retransmits pkt on receipt of NAK
- › new mechanisms in rdt2.0 (beyond rdt1.0):
 - error detection
 - feedback: control msgs (ACK,NAK) from receiver to sender







what happens if ACK/NAK corrupted?

- › sender does not know what happened at receiver!
- › cannot just retransmit, possible duplicate

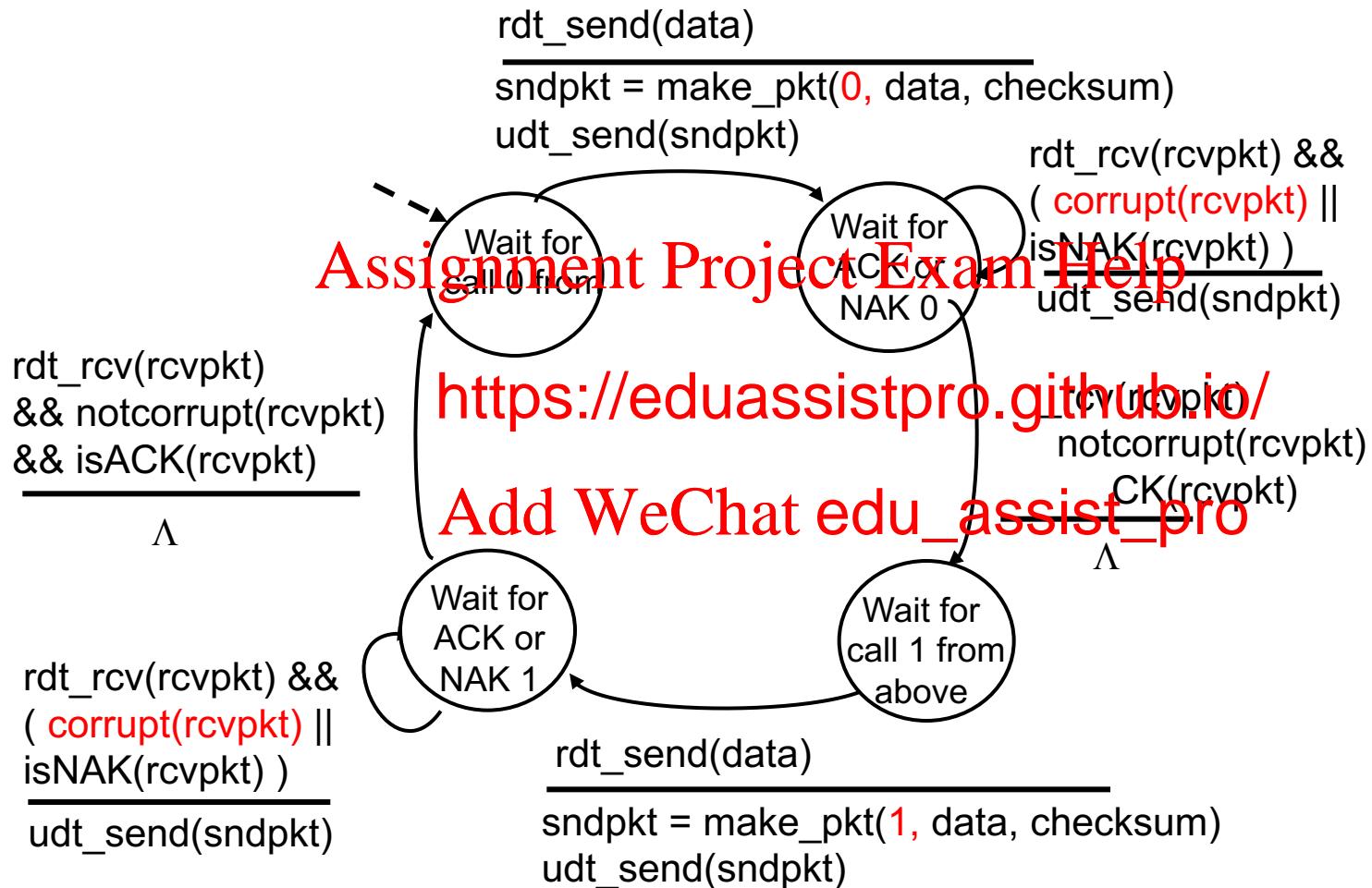
handling duplicates:

- › sender retransmits current pkt if ACK/NAK corrupted
- › sender adds sequence number

<https://eduassistpro.github.io/>
ards (does no
licate pkt
Add WeChat [edu_assist_pro](#)

stop and wait
sender sends one packet,
then waits for receiver
response

rdt2.1: sender, handles garbled ACK/NAKs



rdt2.1: receiver, handles garbled ACK/NAKs

rdt_rcv(rcvpkt) && (corrupt(rcvpkt)

 sndpkt = make_pkt(NAK, chksum)
 udt_send(sndpkt)

rdt_rcv(rcvpkt) &&
 not corrupt(rcvpkt) &&
 has_seq1(rcvpkt)

 sndpkt = make_pkt(ACK, chksum)
 udt_send(sndpkt)

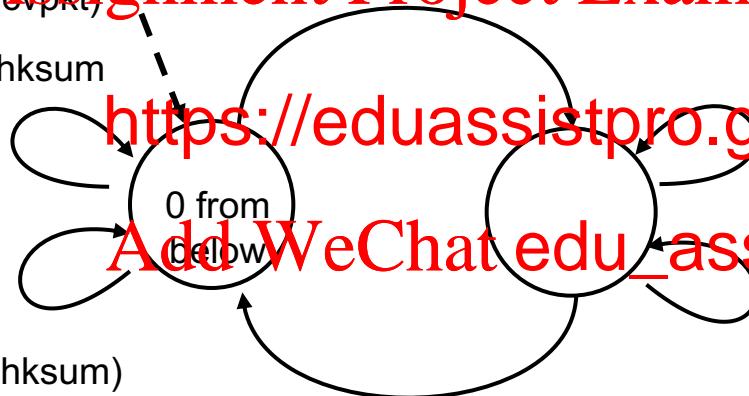
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
 && has_seq0(rcvpkt)

 extract(rcvpkt,data)
 deliver_data(data)
 sndpkt = make_pkt(ACK, chksum)
 udt_send(sndpkt)

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
 && has_seq1(rcvpkt)

 extract(rcvpkt,data)
 deliver_data(data)
 sndpkt = make_pkt(ACK, chksum)
 udt_send(sndpkt)

Assignment Project Exam Help



rdt_rcv(rcvpkt) && (corrupt(rcvpkt)

 sndpkt = make_pkt(NAK, chksum)
 udt_send(sndpkt)

rdt_rcv(rcvpkt) &&
 not corrupt(rcvpkt) &&
 has_seq0(rcvpkt)

 sndpkt = make_pkt(ACK, chksum)
 udt_send(sndpkt)



sender:

- › seq # added to pkt
- › two seq. #'s (0, suffice.
- › must check if received ACK/NAK corrupted
- › twice as many states

- state must “remember” whether “expected” pkt should have seq # of 0 or 1

receiver:

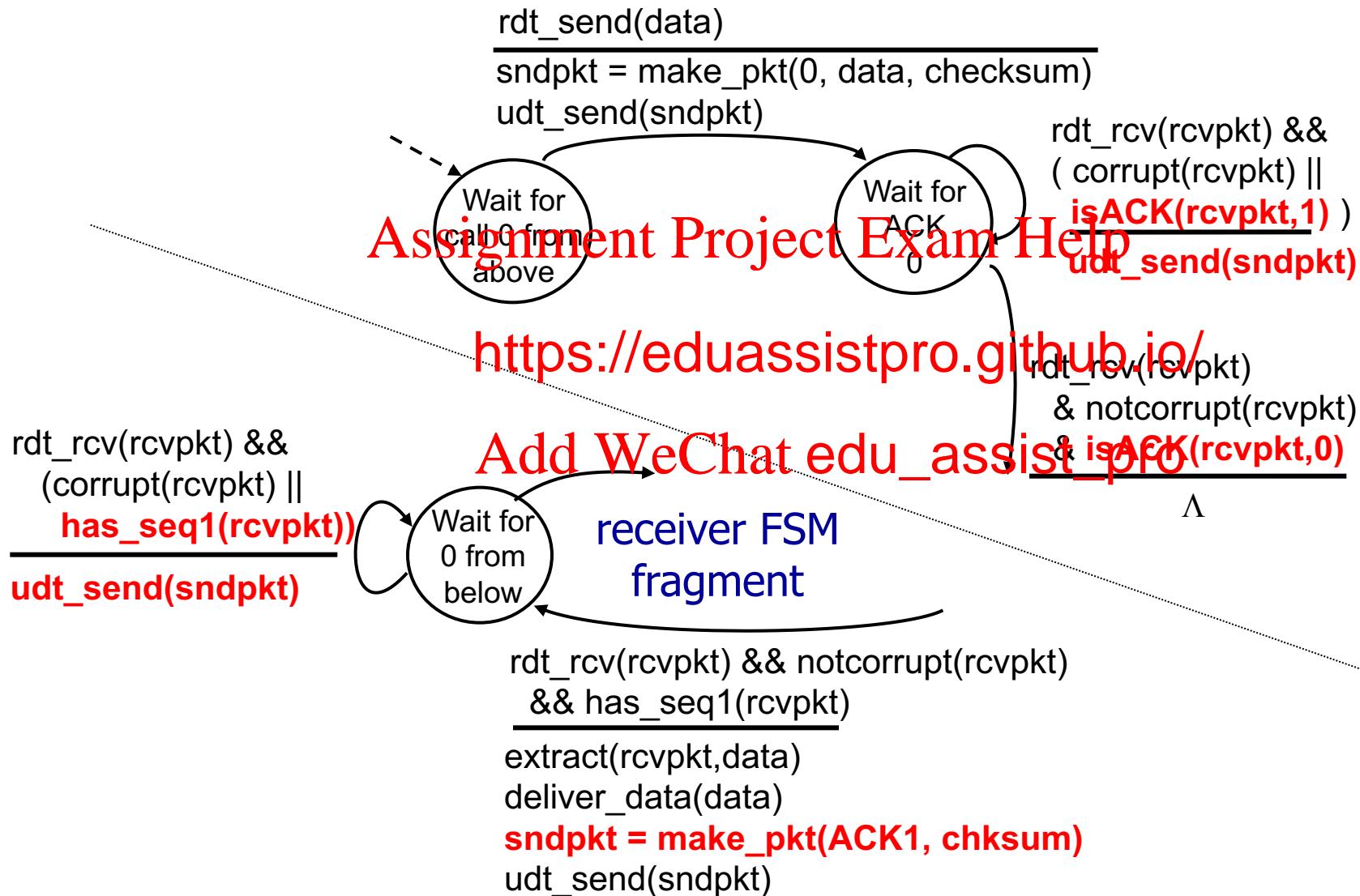
- › must check if received packet is duplicate

<https://eduassistpro.github.io/>

ates whether 0 or ed pkt seq #

Add WeChat edu_assist_pro

- › same functionality as rdt2.1, using ACKs only
- › instead of NAK, receiver sends ACK for last pkt received
- Assignment Project Exam Help
 - receiver must ex <https://eduassistpro.github.io/> being ACKed
- › "unexpected" A in same action as NAK: retransmit current pkt



new assumption: underlying channel can also lose packets (data, ACKs)

- checksum, seq. # retransmissions help ... but not e

approach: sender waits “reasonable” amount of time for ACK

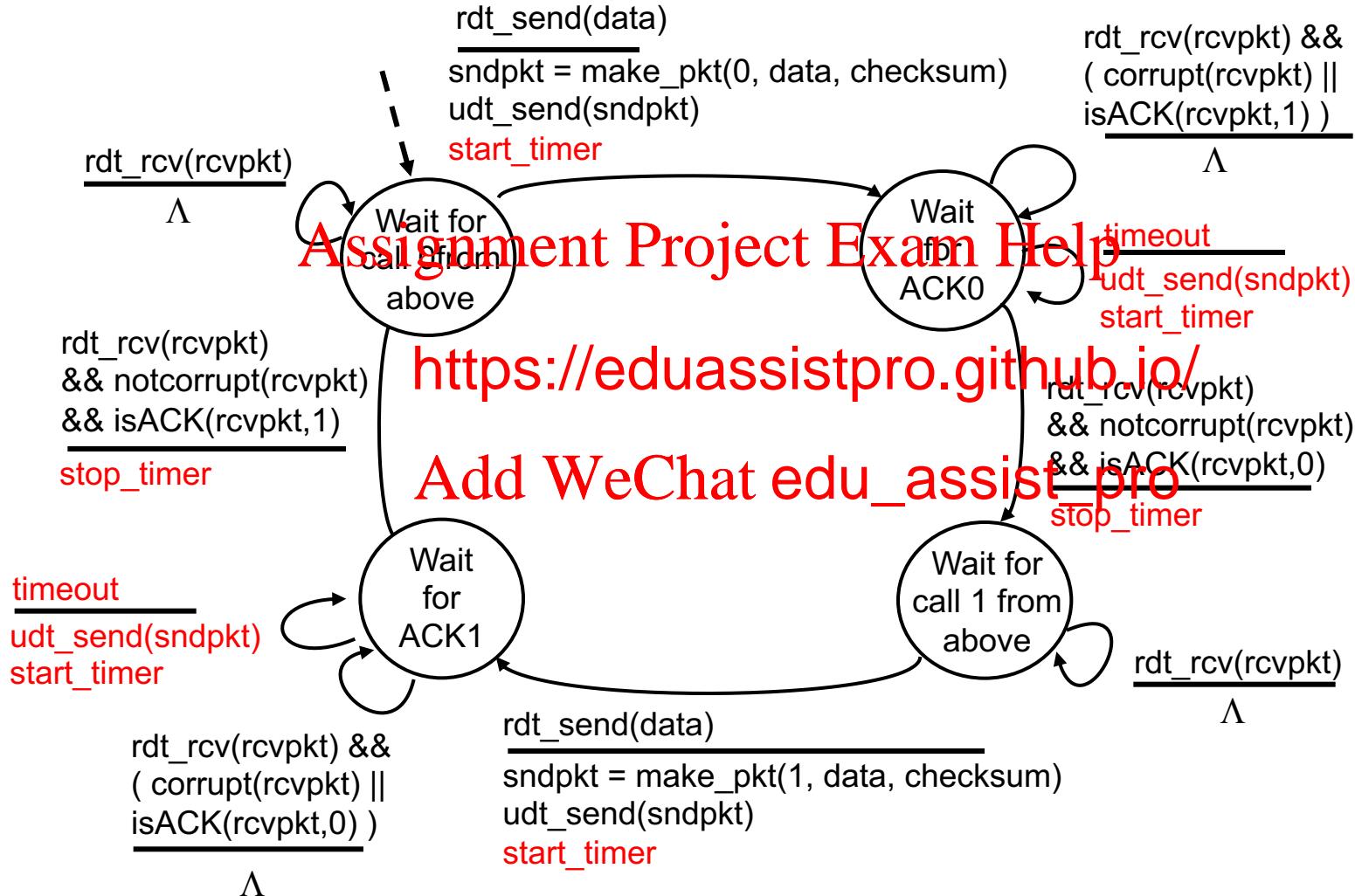
- › retransmits if no ACK received in

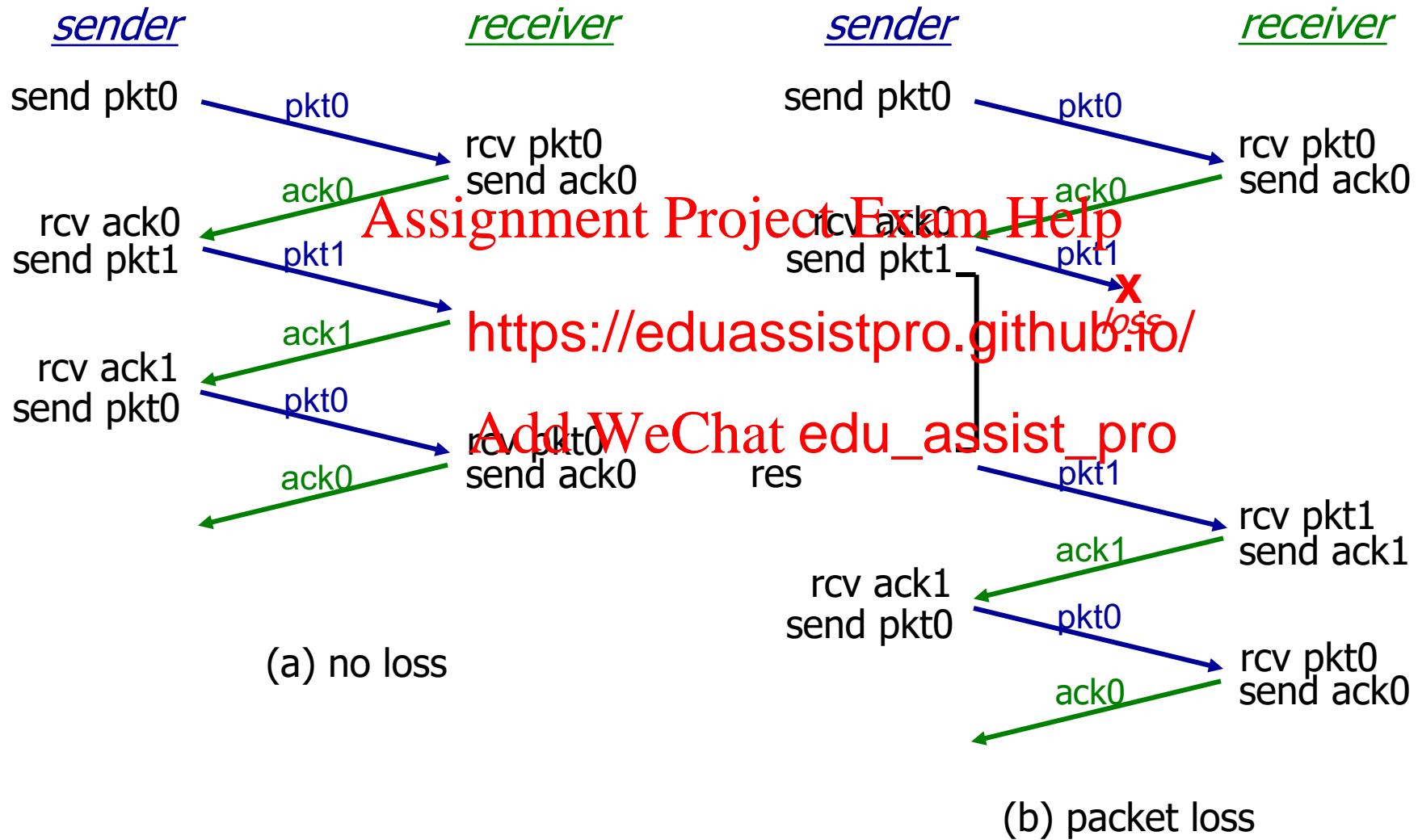
<https://eduassistpro.github.io/>

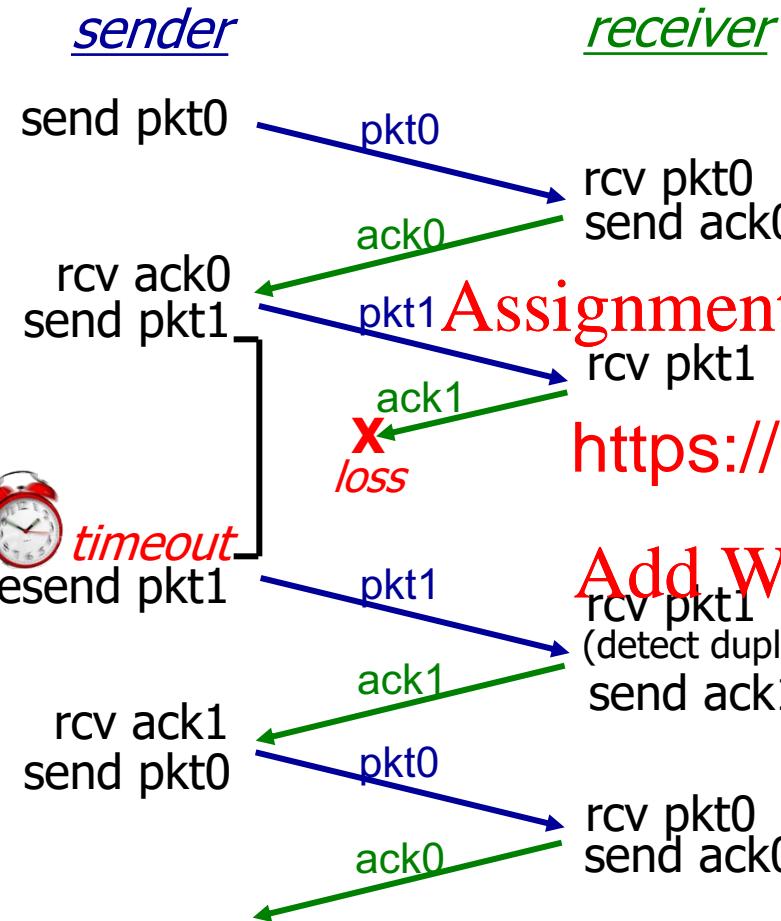
Add WeChat `edu_assist_pro`

will be duplicate, but seq. #'s already handles this

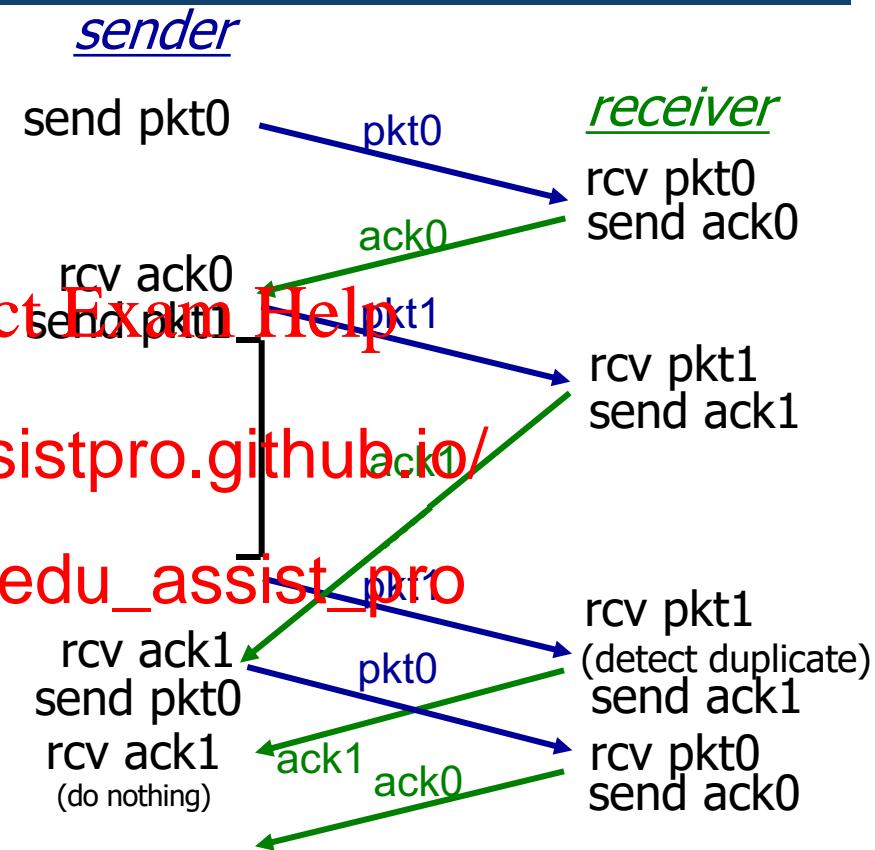
- receiver must specify seq # of pkt being ACKed
- › requires countdown timer







(c) ACK loss



(d) premature timeout/ delayed ACK

- › rdt3.0 is correct, but performance stinks
- › e.g.: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

$$D_{trans} = \frac{8000 \text{ bits}}{1 \text{ Gbps}} = 8 \text{ microseconds}$$

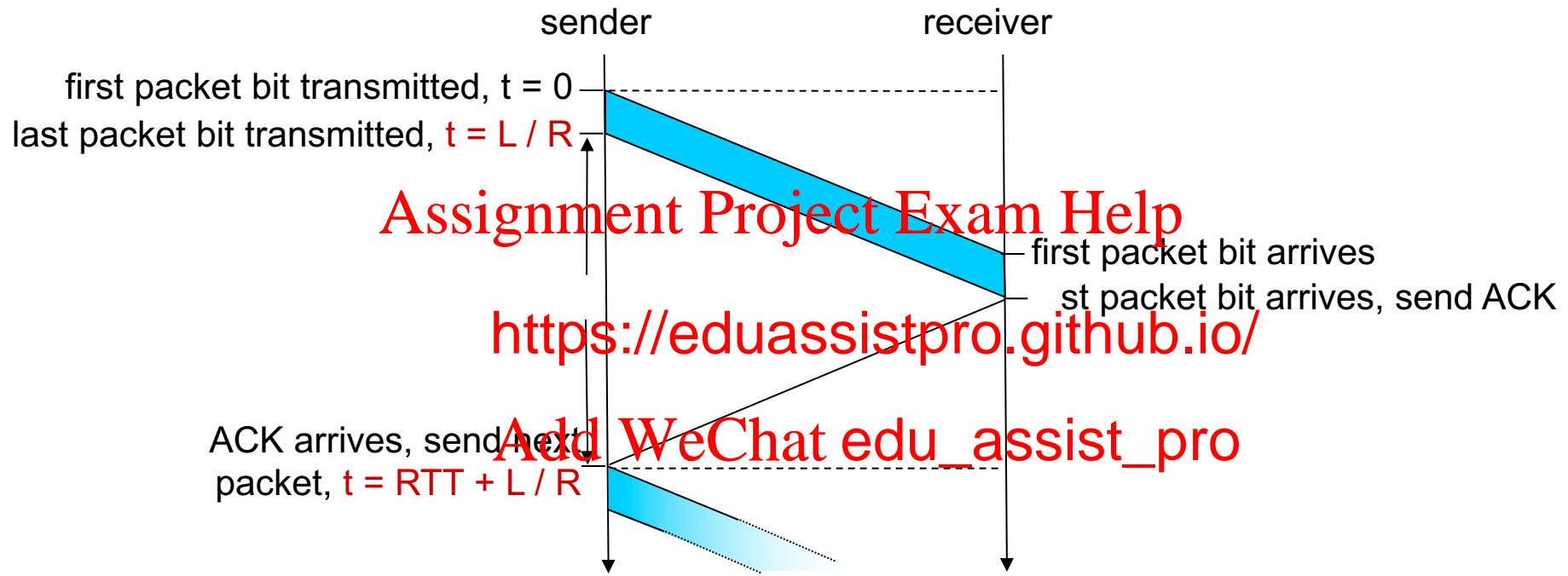
<https://eduassistpro.github.io/>

- U_{sender} : *utilization* – fraction of time busy sending

Add WeChat edu_assist_pro

$$U_{\text{sender}} = \frac{L/R}{RTT + L/R} = \frac{8000}{30.008} = 0.00027$$

- if RTT=30 msec, 1KB pkt every 30 msec: 33kB/sec thruput over 1 Gbps link
- ❖ network protocol limits use of physical resources!



$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts

- range of sequence numbers must be increased
- buffering at send

Assignment Project Exam Help

<https://eduassistpro.github.io/>

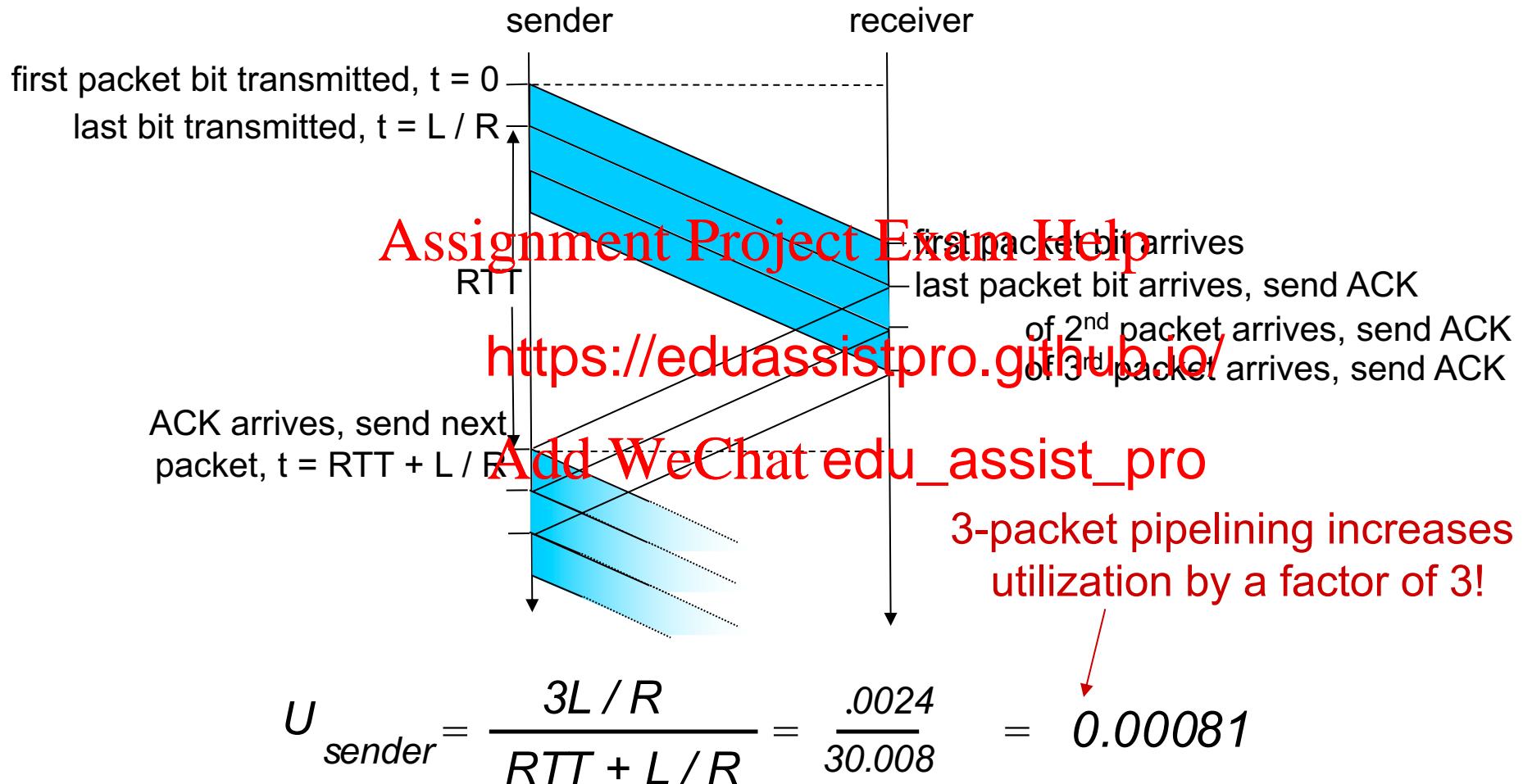


Add WeChat `edu_assist_pro`



- › two generic forms of pipelined protocols: *go-Back-N*, *selective repeat*

Pipelining: increased utilization



Go-back-N:

- › sender can have up to N unacked packets in pipeline
- › receiver only sends **cumulative ack**
 - does not ack packet if there is a gap
- › sender has timer for oldest unacked packet
 - when timer expires, retransmit *all* unacked packets

Selective Repeat:

- › sender can have up to N unacked packets in pipeline
- › receiver sends **individual ack** packet
- › Add WeChat `edu_assist_pro` contains timer for unacked packet
 - when timer expires, retransmit only that unacked packet

- › “window” of up to N, consecutive unacked pkts allowed

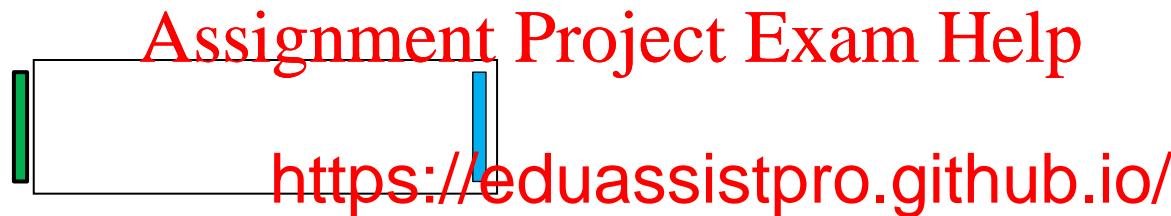
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

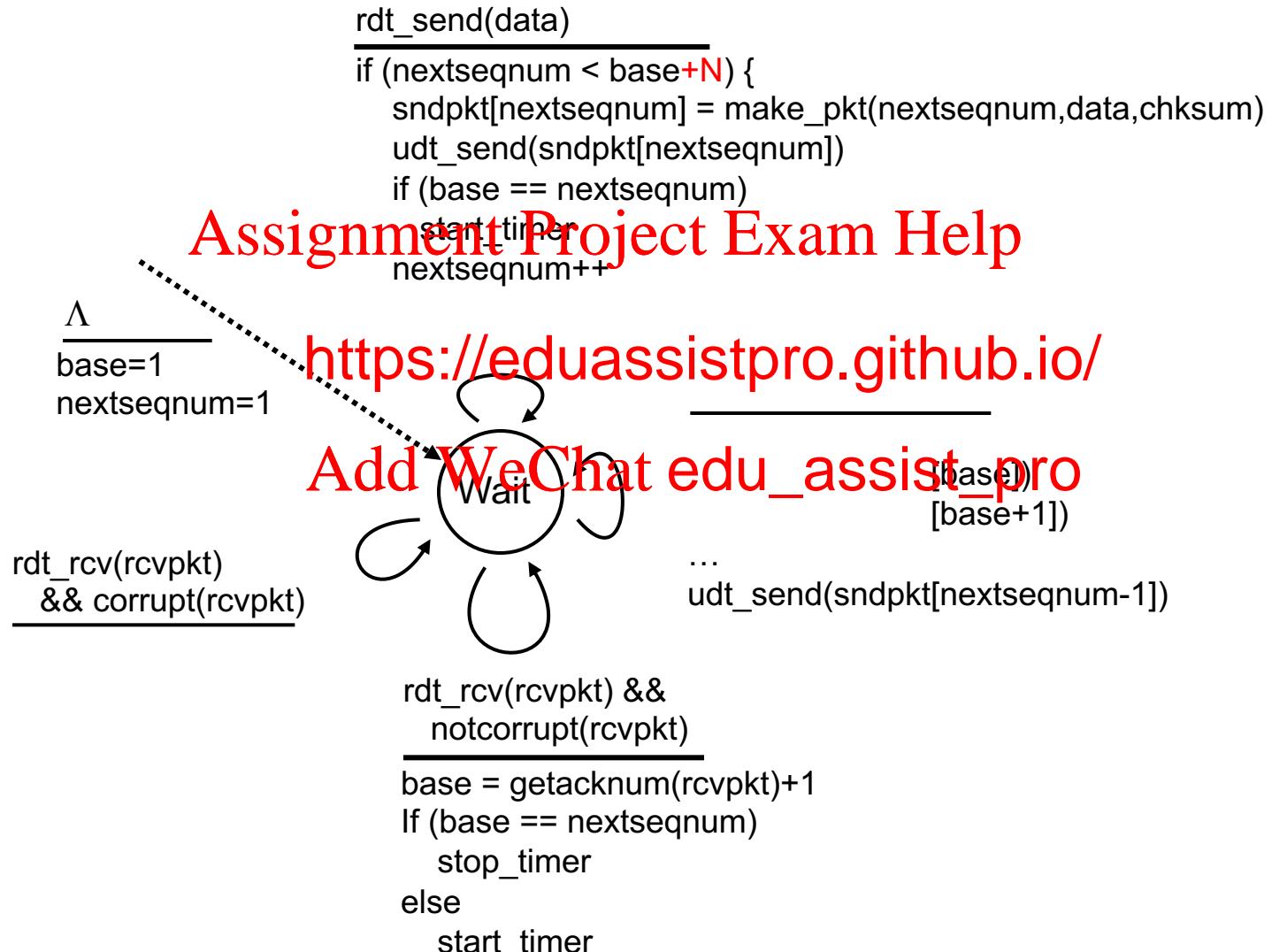
- ❖ ACK(n):ACKs all pkts up to, including seq # n - “*cumulative ACK*”
 - may receive duplicate ACKs (see receiver)
- ❖ timer for oldest in-flight pkt
- ❖ *timeout(n)*: retransmit packet n and all higher seq # pkts in window

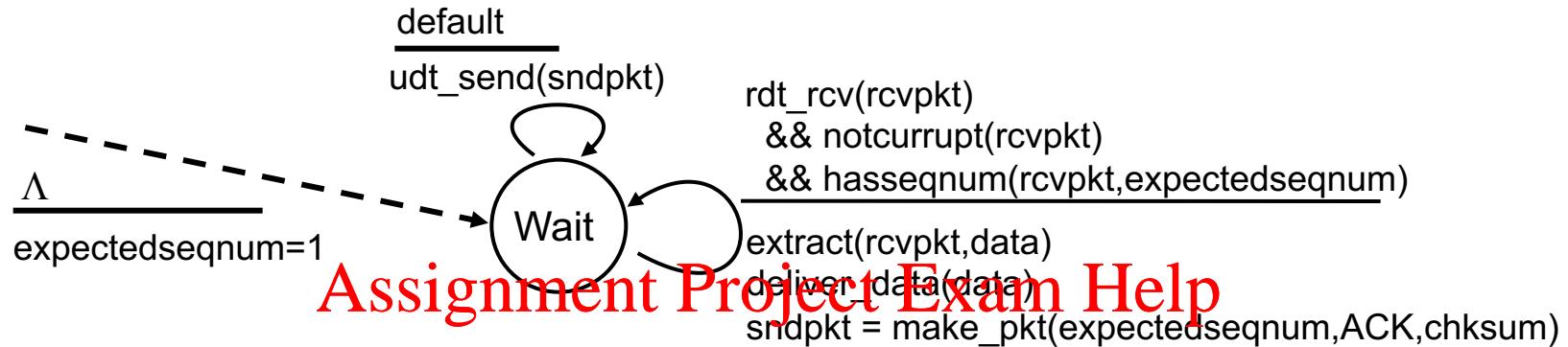
- › “window” of up to N, consecutive unacked pkts allowed



Add WeChat `edu_assist_pro`

- ❖ ACK(n): ACKs all pkts up to, including seq # n - “*cumulative ACK*”
 - may receive duplicate ACKs (see receiver)
- ❖ timer for oldest in-flight pkt
- ❖ $\text{timeout}(n)$: retransmit packet n and all higher seq # pkts in window





Assignment Project Exam Help

<https://eduassistpro.github.io/>

ACK-only: always send ACK
 Add WeChat edu_assist_pro
 ly-received pkt with
 highest *in-order* seq #

- may generate duplicate ACKs
 - need only remember **expectedseqnum**
- › out-of-order pkt:
- discard (don't buffer): *no receiver buffering!*
 - re-ACK pkt with highest in-order seq #

sender window (N=4)

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

sender

send pkt0
send pkt1
send pkt2
send pkt3

(wait)

receiver

receive pkt0, send ack0
receive pkt1, send ack1

receive pkt3, discard,
(re)send ack1

receive pkt4, discard,
(re)send ack1
receive pkt5, discard,
(re)send ack1

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

rcv ack0, s
rcv ack1, s

<https://eduassistpro.github.io/>

ignore duplicate ACK



pkt 2 timeout

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

send pkt2
send pkt3
send pkt4
send pkt5

rcv pkt2, deliver, send ack2
rcv pkt3, deliver, send ack3
rcv pkt4, deliver, send ack4
rcv pkt5, deliver, send ack5



- › receiver *individually* acknowledges all correctly received pkts
 - buffers pkts as needed, for eventual in-order delivery to upper layer
- › sender only re <https://eduassistpro.github.io/> received
 - sender timer for each unACKed
- › sender window
- › receiver window

Selective repeat: sender, receiver windows



Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro



sender**data from above:**

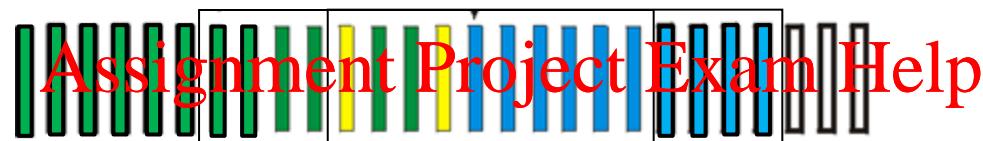
- › if next available seq # in window, send pkt
- time out(n):
 - › resend pkt n, res
 - ACK(n) in [sendbase,sendbase+N-1]:
 - › mark pkt n as received
 - › if n is smallest unACKed pkt, advance window base to next unACKed seq #

receiver**pkt n in [rcvbase, rcvbase+N-1]**

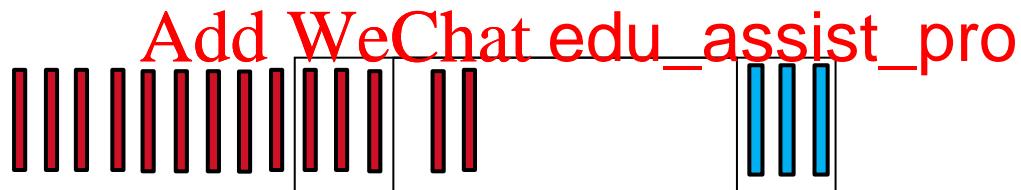
- ❖ send ACK(n)
- out-of-order: buffer
 - der: deliver (also buffered, in-order window to t-yet-received pkt [rcvbase-N,rcvbase-1])
- ACK(n)
- otherwise:
 - ❖ ignore

<https://eduassistpro.github.io/>

Add WeChat **edu_assist_pro**



<https://eduassistpro.github.io/>



Selective repeat in action

sender window (N=4)

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

sender

send pkt0
send pkt1
send pkt2
send pkt3

Assignment Project Exam Help
(wait)

receiver

receive pkt0, send ack0
receive pkt1, send ack1

receive pkt3, buffer,
send ack3

receive pkt4, buffer,
send ack4
receive pkt5, buffer,
send ack5

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

0	1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---	---

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

pkt 2 timeout

send pkt2

record ack4 arrived

record ack5 arrived

Q: what happens when ack2 arrives?

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8