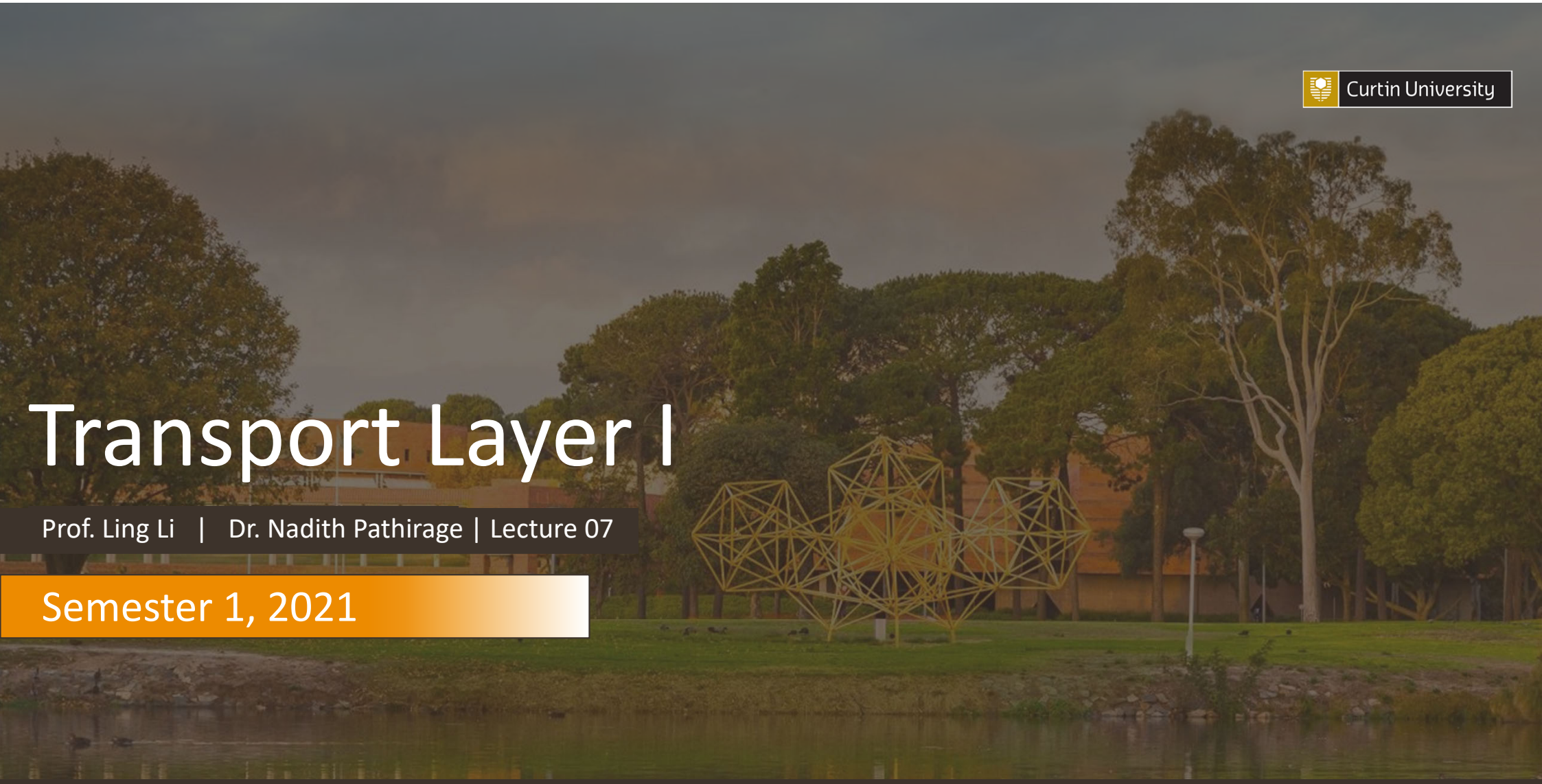
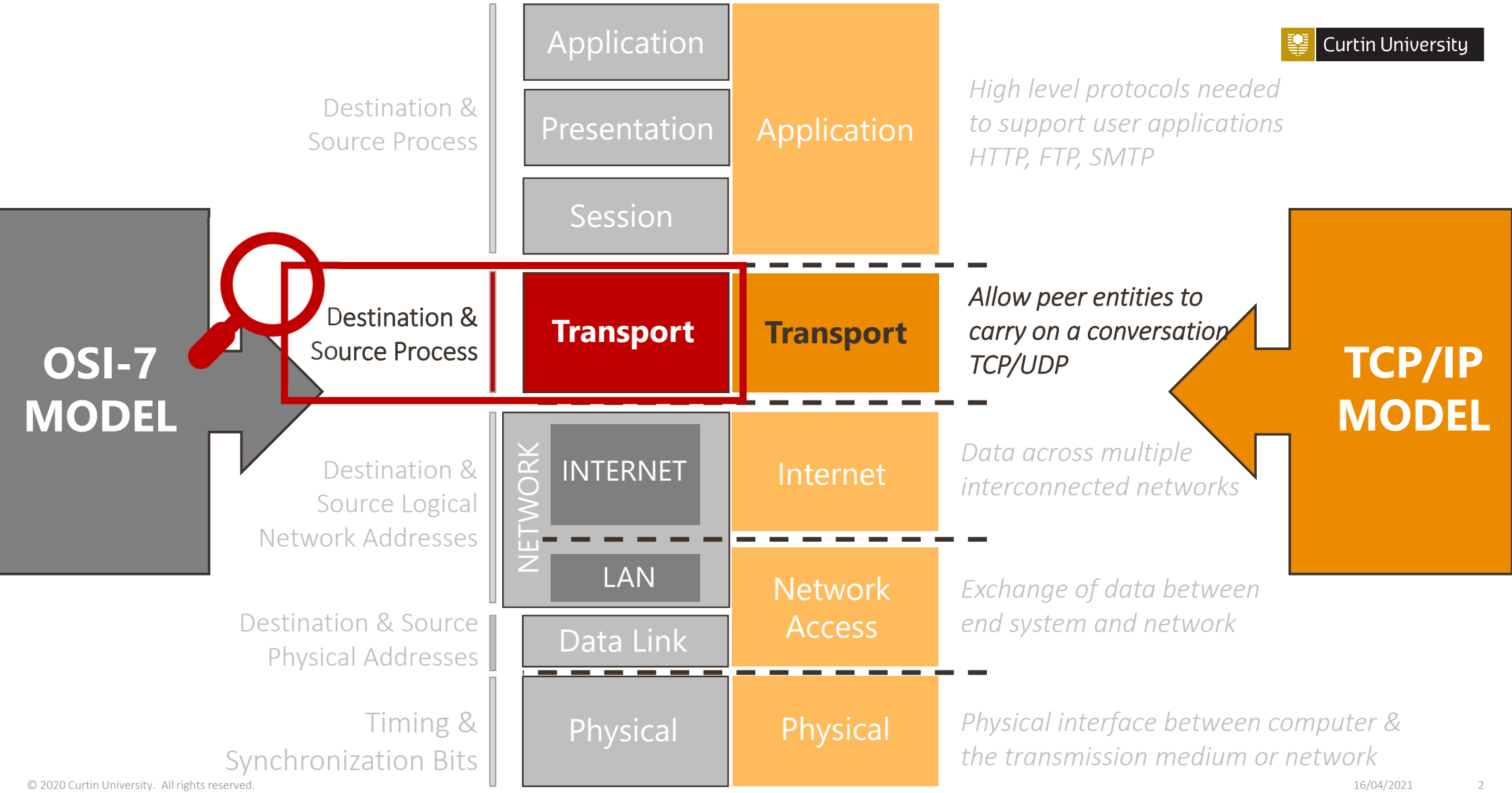


Curtin University

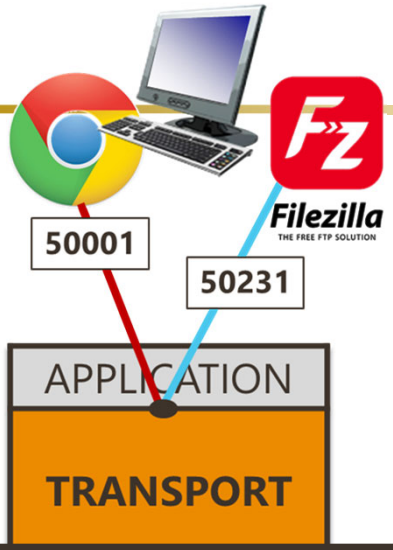
Transport Layer I

Prof. Ling Li | Dr. Nadith Pathirage | Lecture 07

Semester 1, 2021



Transport Layer

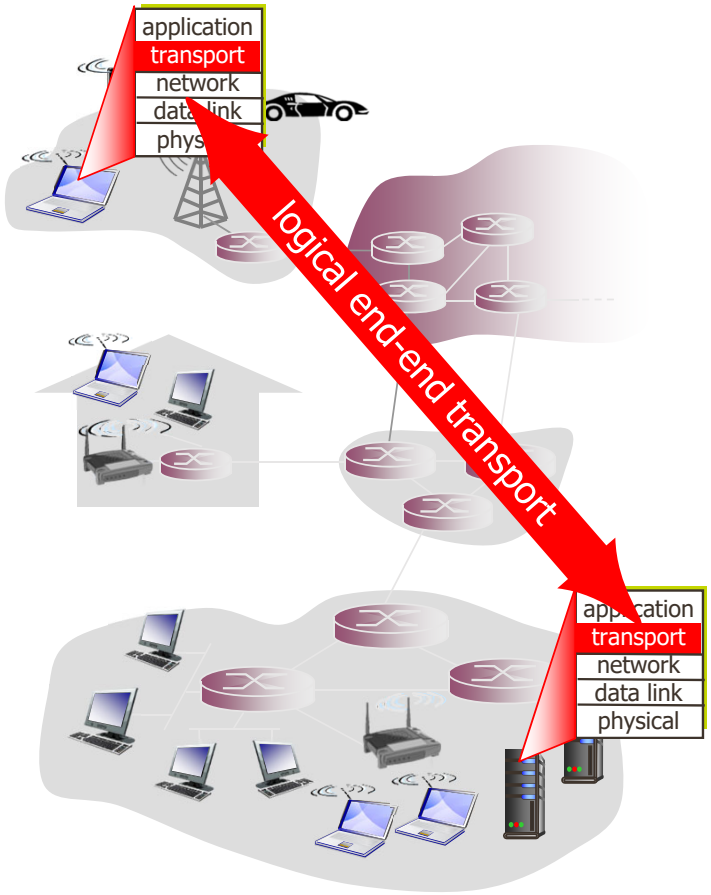


Transport layer provides communication services for a **process** in a host **to** a **process** in another host

Transport Layer

Curtin University

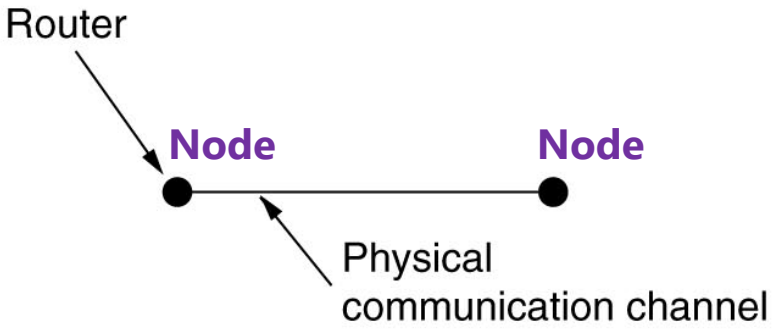
- **Heart** of whole protocol hierarchy
- **Reliable** data transmission
- **Cost-effective** data transport
- **Independent** of physical network



Transport Layer – cont.

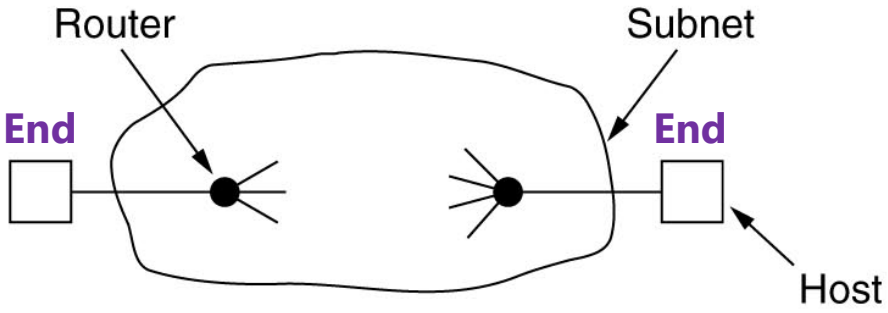
Curtin University

- Provides **end-to-end communication** for individual applications



(a)

Data Link Layer



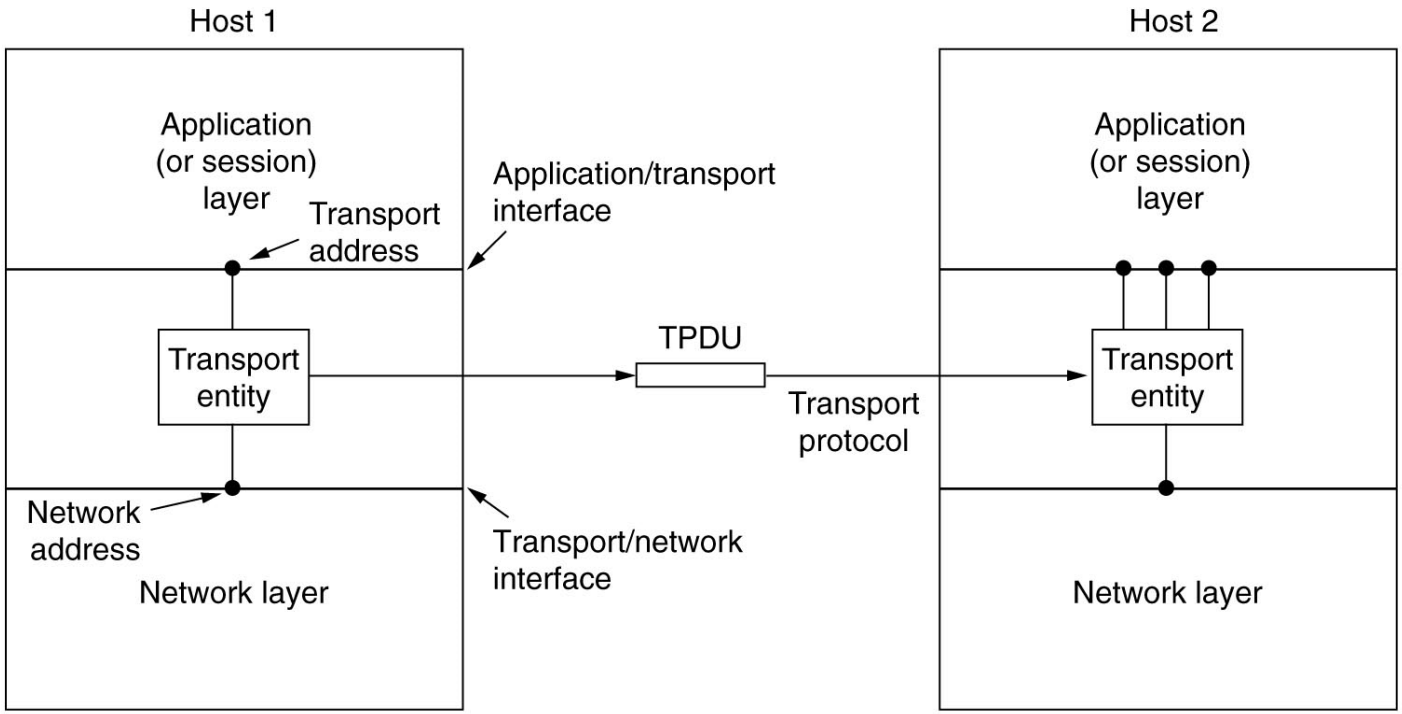
(b)

Network Layer

Transport Entity

Hardware and/or software within the transport layer

- *Located in:*
  - ✓ OS **kernel** *or*
  - ✓ Separated **user process** *or*
  - ✓ **Lib** package bound into network application *or*
  - ✓ Conceivably on the network interface card **(NIC)**



Why Transport layer?

Curtin University

- Transport layer entity on user's machines, while network layers mostly runs on routers
- To be more reliable than the underlying network service
- **Transport layer primitives** can be implemented as calls to library procedures in order to make them independent of network service primitives

Curtin University

Transport Layer
Elements

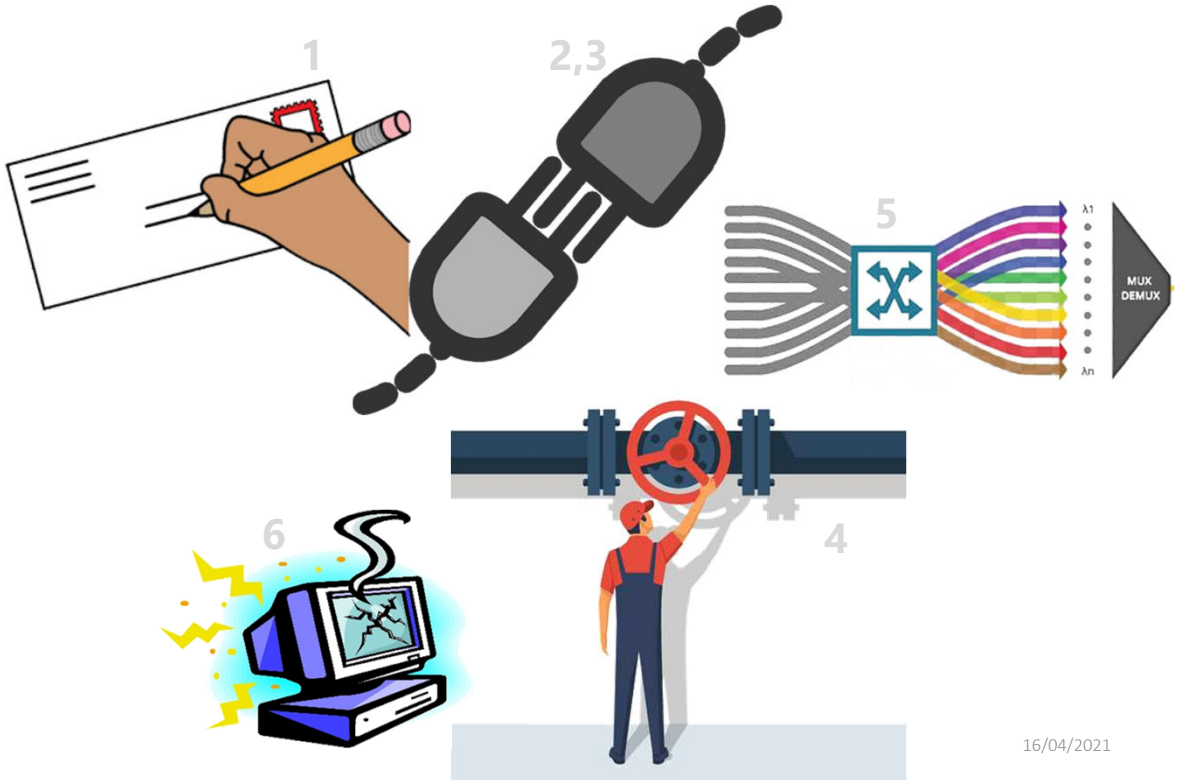
- Addressing
- Connection Establishment
- Connection Release
- Flow Control and Buffering
- Multiplexing
- Crash Recovery

Transport Layer Elements

Curtin University

- **Transport Protocols ~= Data Link Protocols**
  - ✓ *Error control*
  - ✓ *Sequencing*
  - ✓ *Flow control and other issues.*

- *differences in:*
  1. **Addressing**
  2. **Connection Establishment**
  3. **Connection Release**
  4. **Flow Control & Buffering**
  5. **Multiplexing**
  6. **Crash Recovery**



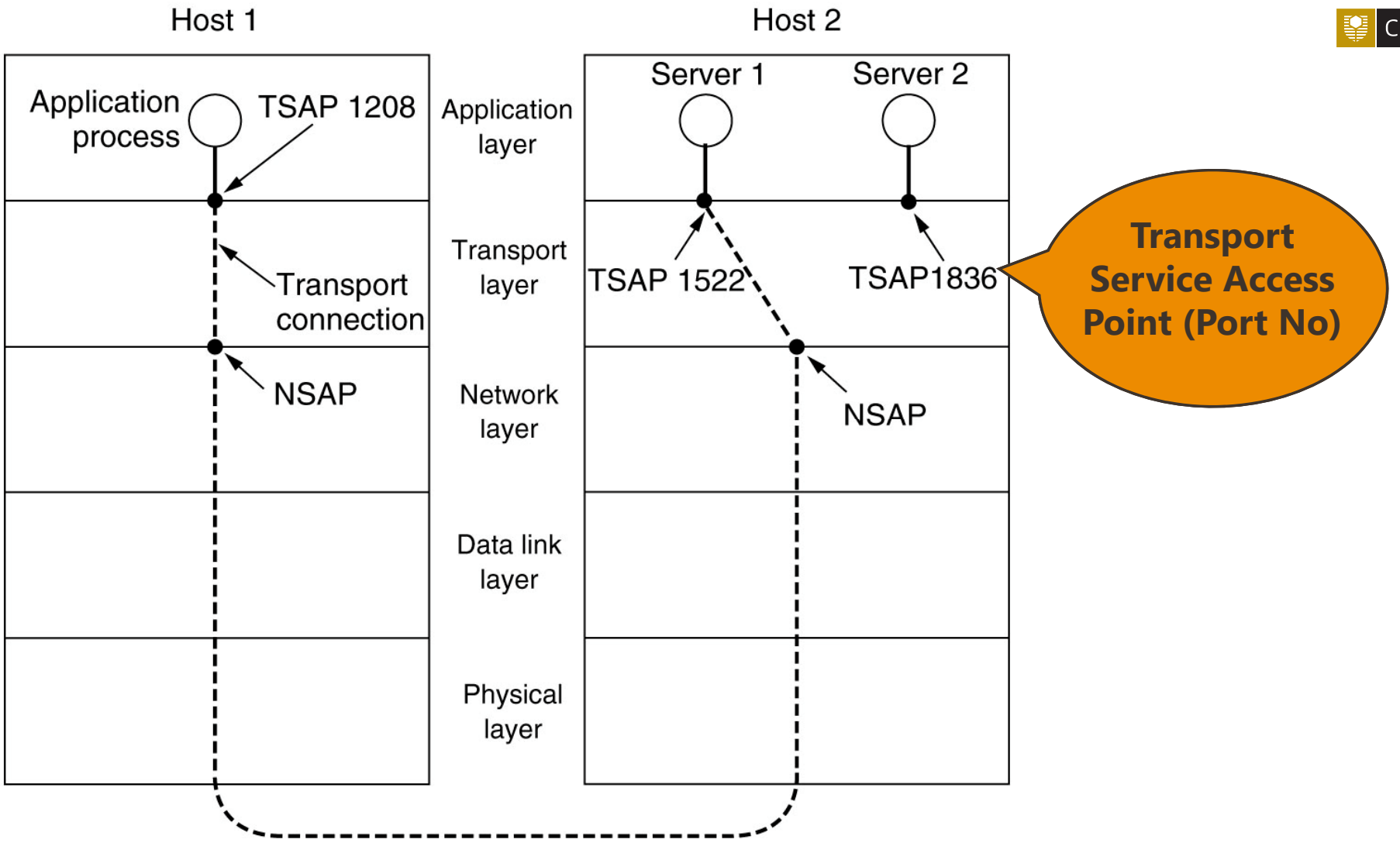
1. Addressing

Curtin University

Which application (server) on a remote host **to connect**

✓ *Assign a* **port number** *to Application !*

How to determine the **port number** of a particular application



1. Addressing: TSAP

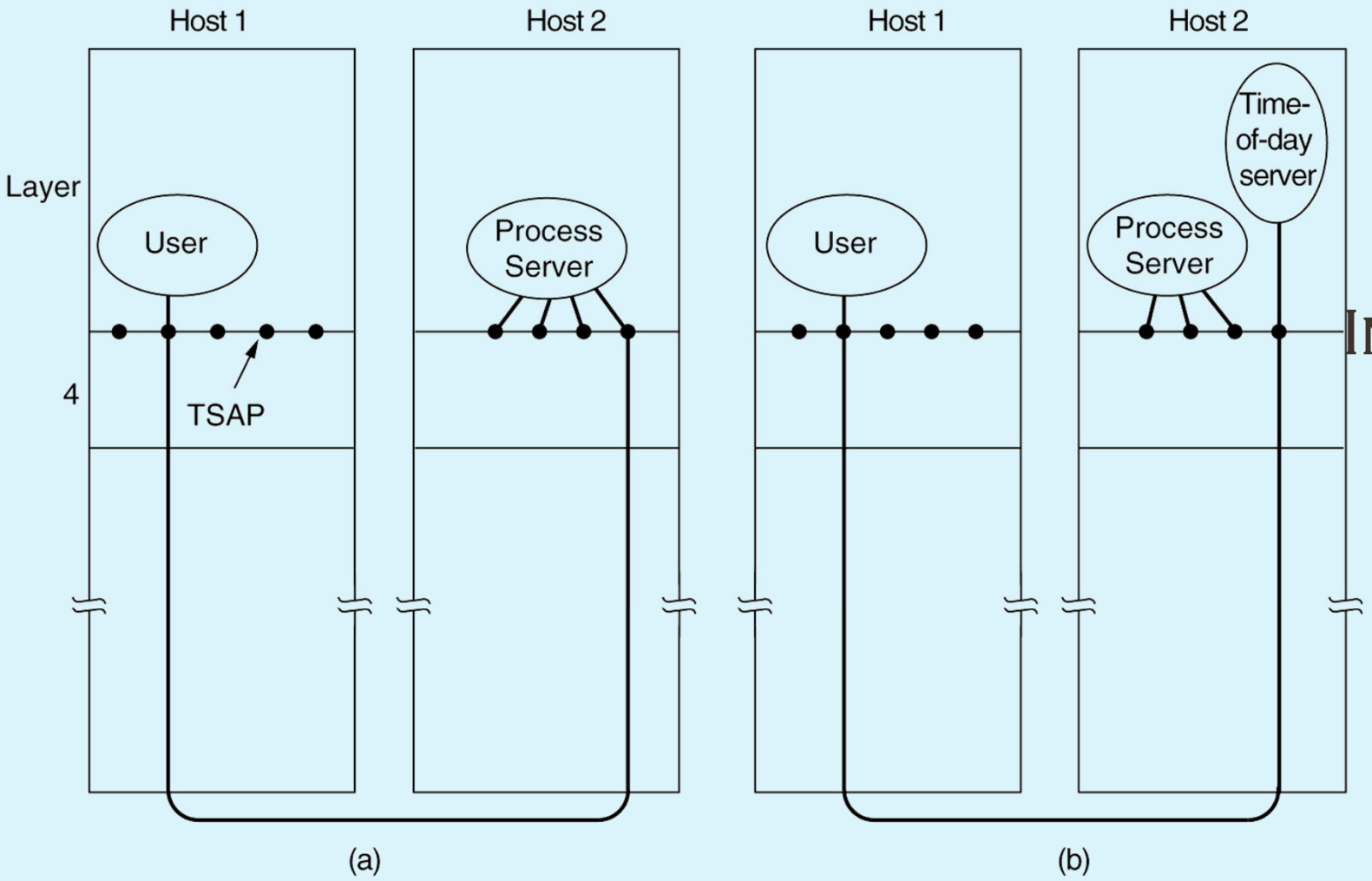
Curtin University

- Popular applications use the same TSAP address (permanently) on every host
  - **well-known ports**

IMPLEMENTATION OPTIONS

- Others use:
  - **Initial Connection Protocol**
    - a special Process Server acts as a **proxy** which listen to a set of less heavily used servers at the same time

  - **Directory Server**
    - a.k.a. Name Server
    - Remote user connect to a well known TSAP/port address to resolves symbolic service names into TSAPs
    - Client break connection with the directory server and make a new connection with the required service on the TSAP/port number supplied



Example initial connection protocol: a user process in host 1 establishing a connection with a time-of-day server in host 2 (Tanenbaum)

Implementation Options

1

Initial Connection Protocol

Complete TCP address (Socket)

Curtin University

<IP>: <Port> *or* <NSAP>:<TSAP>

i.e. 192.168.10.5:80

- Port Numbers Ranges [RFC768]:

Well Known Ports: 0 - 1023

Assigned by the IANA (Internet Assigned Numbers Authority)

Registered Ports: 1024 - 49151

Dynamic / Private Ports: 49152 - 65535

IANA registered, for the convenience of Internet Community

*For the complete list: www.iana.org/assignments/port-numbers

WELL-KNOWN PORTS

50001

50231

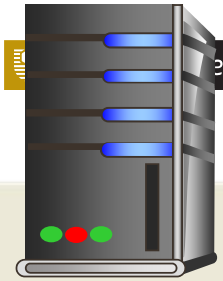
APPLICATION

TRANSPORT

NETWORK

PHYSICAL

<IP>:21 <IP>:80



Server

DHCP NBNS DNS SNTP SNMP Telnet SMTP HTTP FTP

67 137 53 123 161 23 25 80 21

UDP TCP

IP

MAC

PHY

2. Connection Establishment

Curtin University

- Send a Connection Request **(CR)**
- Wait for a Connection Accepted **(CA)**

network can lose, store, and duplicate packets.

Duplicates

Curtin University

- Ensure packets will not live in the network forever

- **Solutions:**
  - ✓ Restrict Packet **lifetime**
  - ✓ Restricted **subnet design**
  - ✓ Putting a **hop counter** in each packet
  - ✓ **Time-stamping** each packet

Duplicates – cont.

Curtin University

- Need to guarantee **not only** that **TPDU** is "dead", **but** also all **acknowledgements**

- Wait a time **T** after a packet has been sent
  - ✓ *Ensure all traces of the packet + its acknowledgements are gone*

T : *some small **multiple** of the true* **maximum packet lifetime**

The multiple is **protocol dependent; makes T longer**

Tomlinson (1975)

Curtin University

Equip each host with a **clock** in the form of a **binary** counter that **continue running even when the host goes down**

- Some part of the counter is used as the **initial sequence no**. when a connection is set up
- Ensure that **two identically numbered TPDUs** are **never outstanding** at the same time
  - ✓ **Sequence space** should be large
- Once both transport entities agreed on the initial sequence number, **any sliding window protocol** can be used for data flow control.

Tomlinson (1975) – cont.

Curtin University

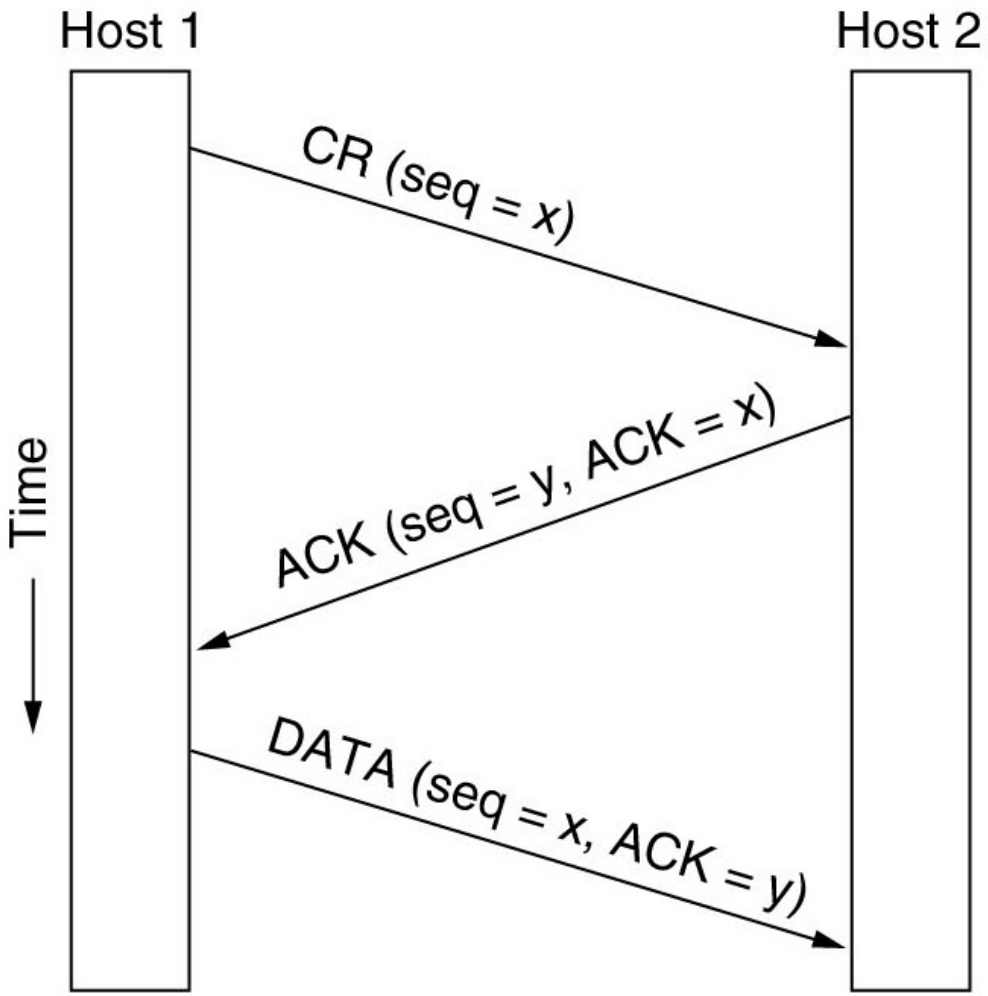
- **what happen when a host crashed?**
  - transport entity's sequence number is lost
  - one solution - require transport entities to be idle for T seconds

    however, T may be large

- **Restriction on** the use of **sequence numbers**
  - Prevent sequence numbers from being used (assigned to new TPDUs) for a time T before their potential use as initial sequence numbers

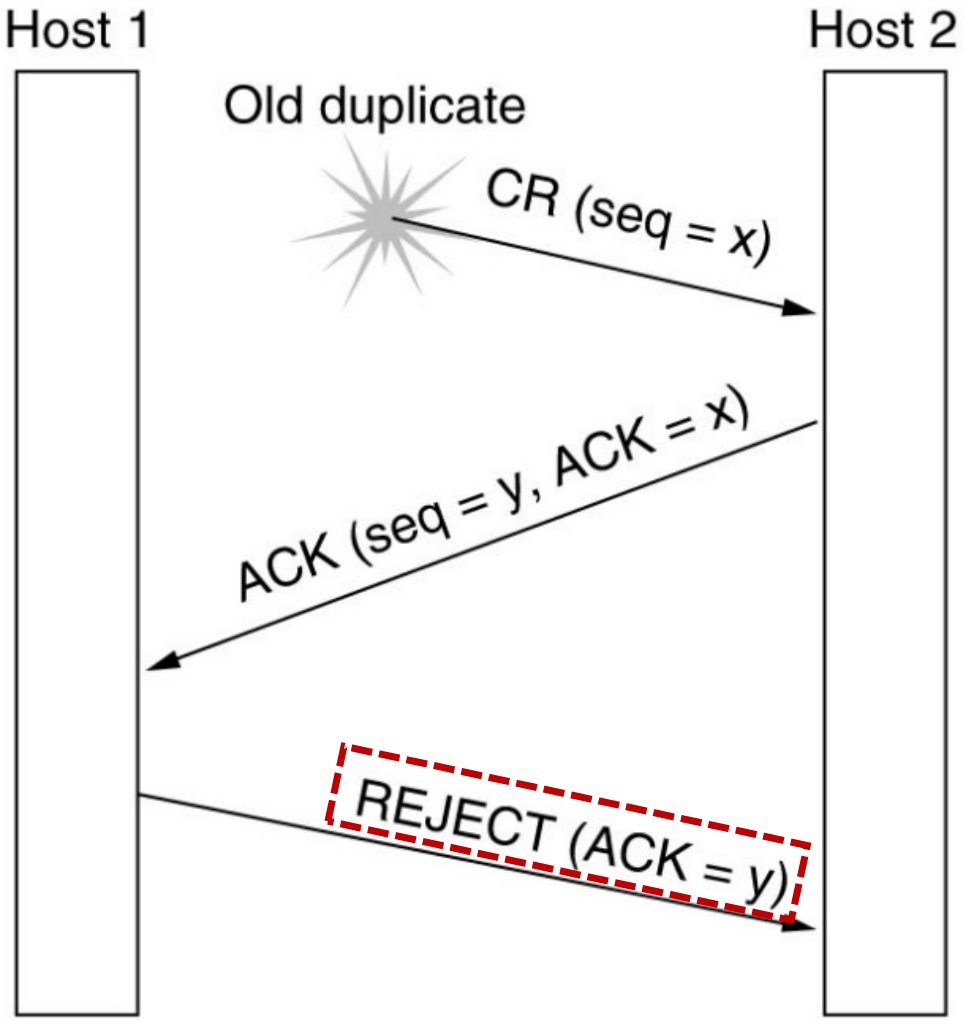
Tomlinson (1975): 3–Way Handshake

Curtin University



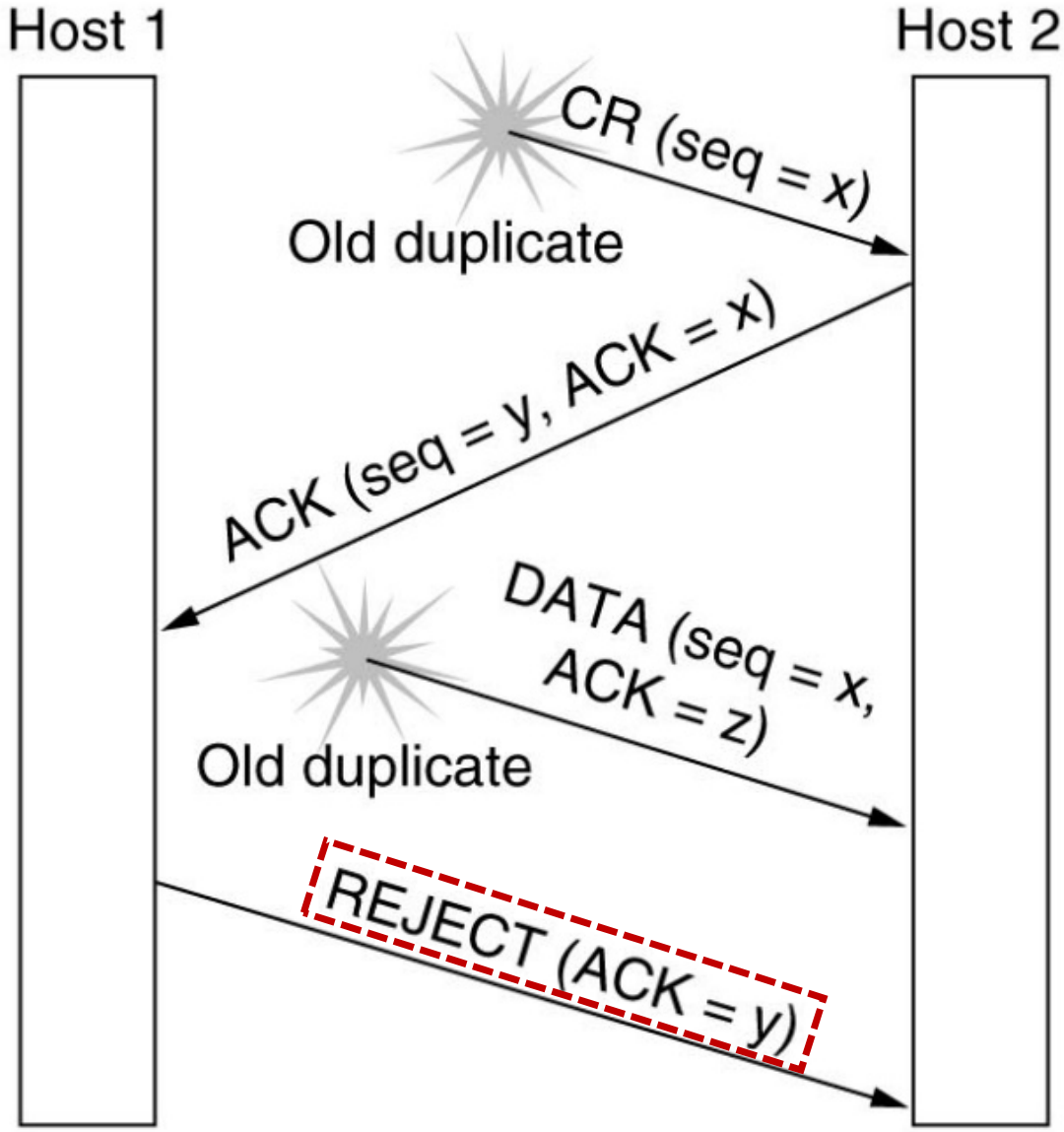
Curtin University

DUPLICATE CR



DUPLICATE CR & DUPLICATE ACK

Initial sequence number previously acknowledged will be the **incorrect** one, and the connection will be **abandoned**

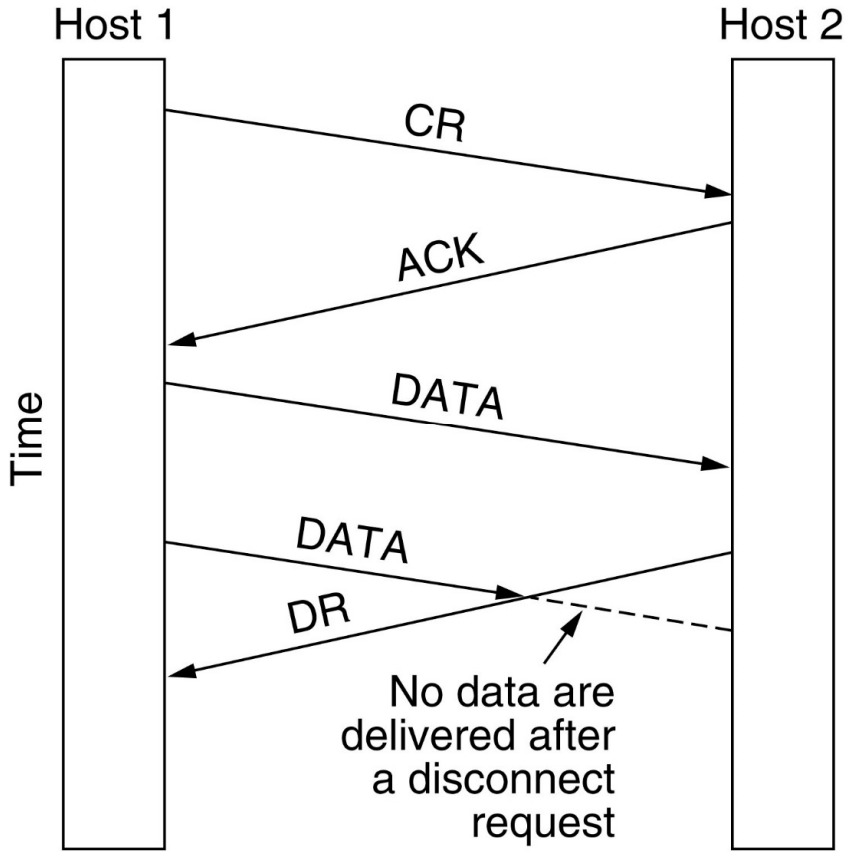


3. Connection Release

Curtin University

1. Asymmetric Release

- ✓ Either user can issue a disconnect
- ✓ Rather abrupt, can result in **loss of data**

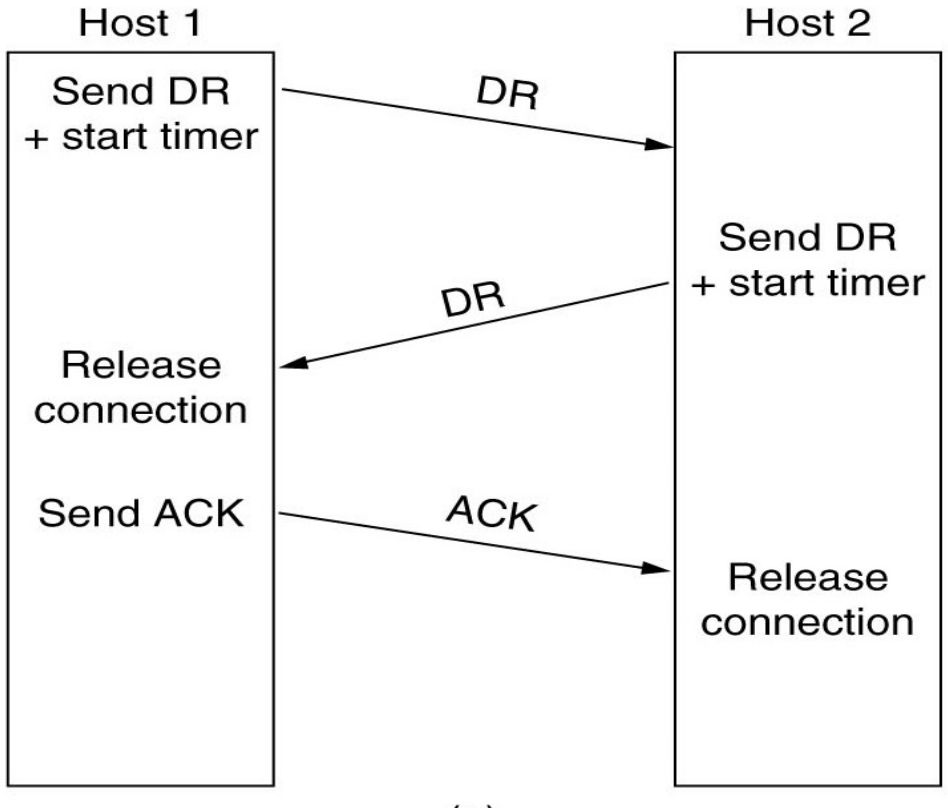


3. Connection Release – cont.

Curtin University

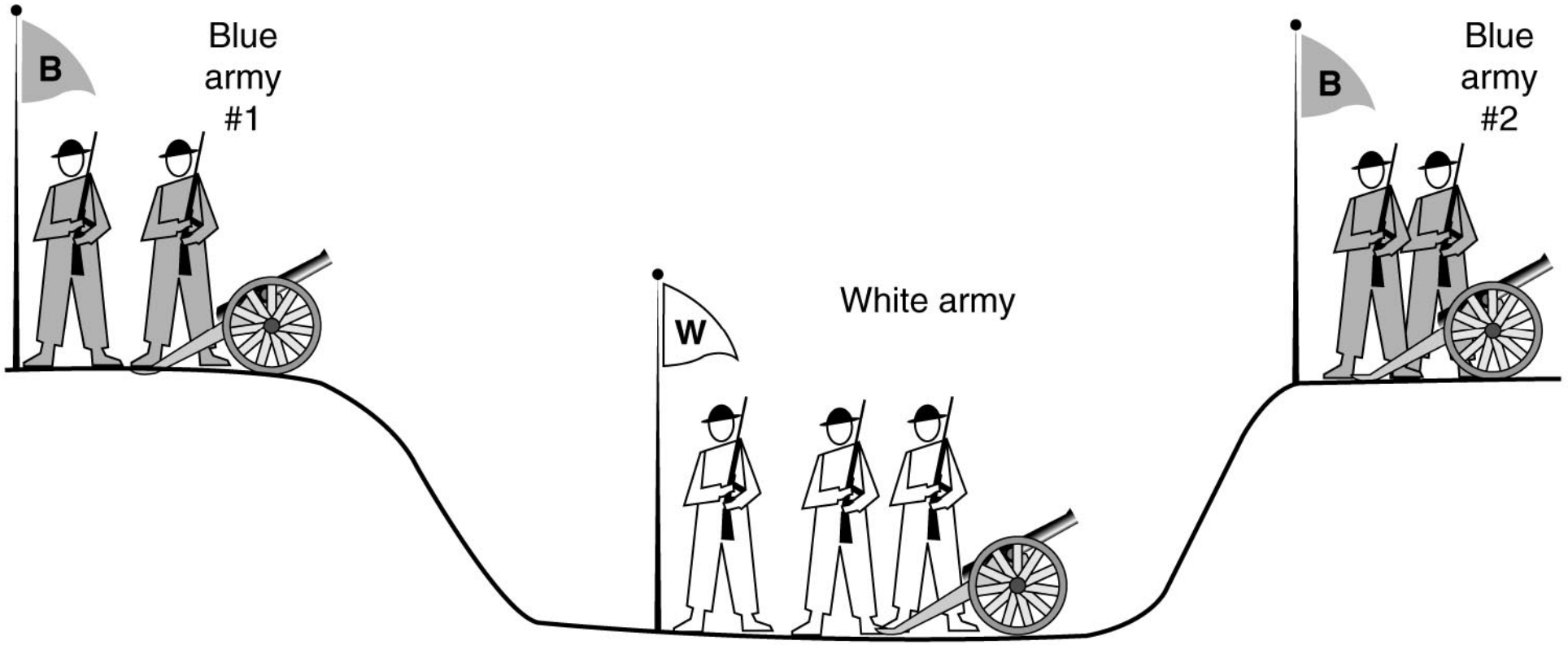
2. Symmetric Release

- ✓ each direction of the connection is closed separately.
- ✓ **Not foolproof !** (two-army problem)
- ✓ Can result in one **half** of a connection remaining **open**



Two Army Problem

Curtin University



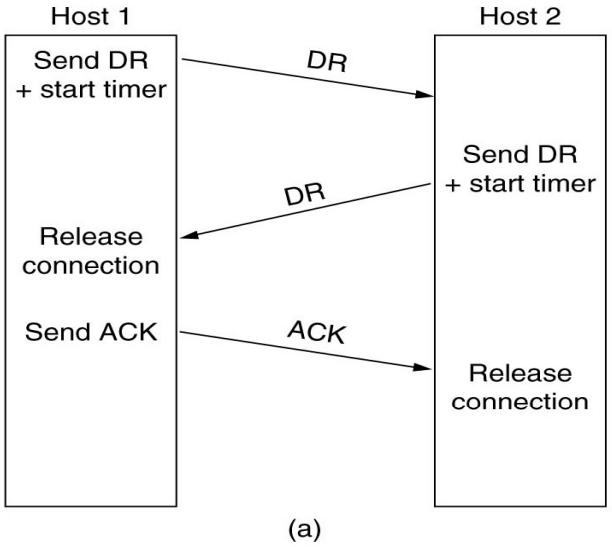
CONN. RELEASE

a) Normal three-way handshake

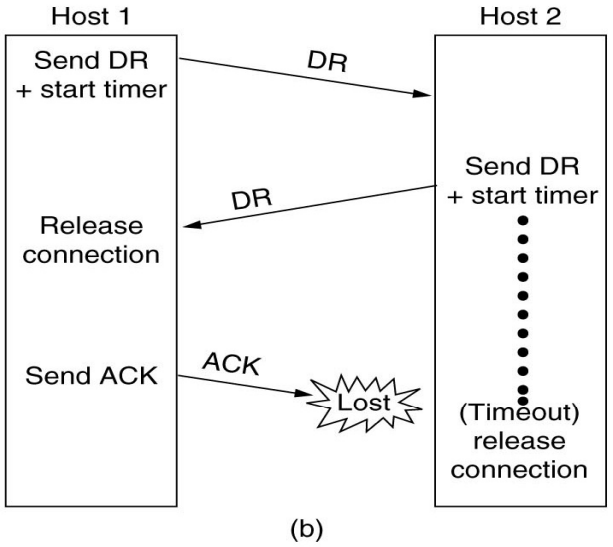
b) Final ACK lost

c) Response lost

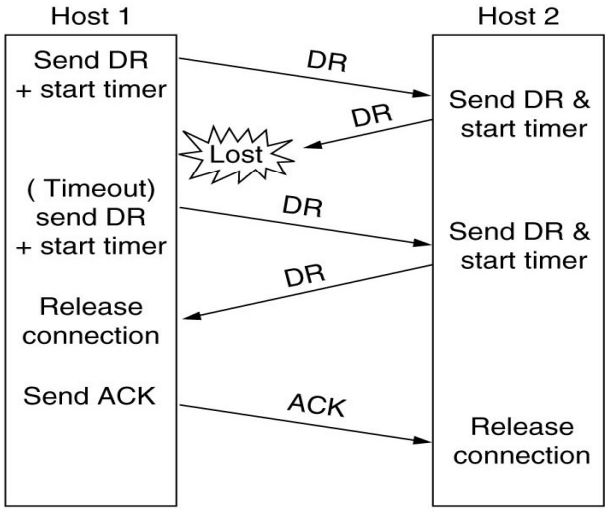
d) Response lost and **subsequent DRs** lost



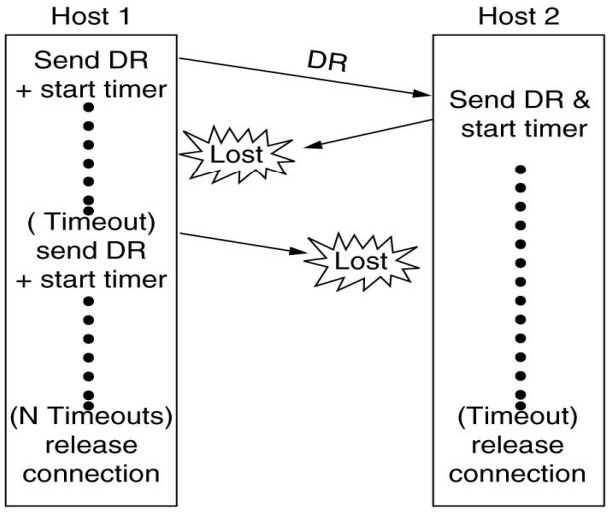
(a)



(b)



(c)

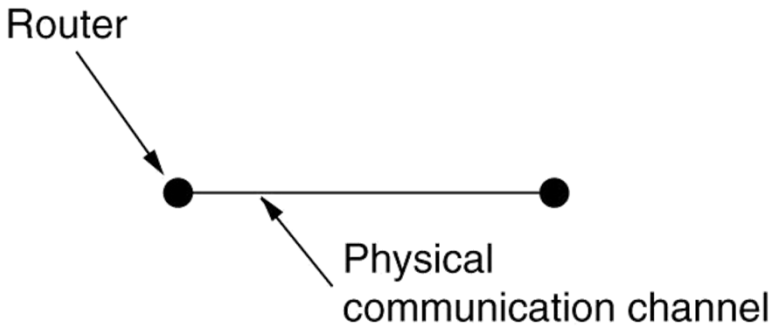


(d)

4. Flow Control & Buffering

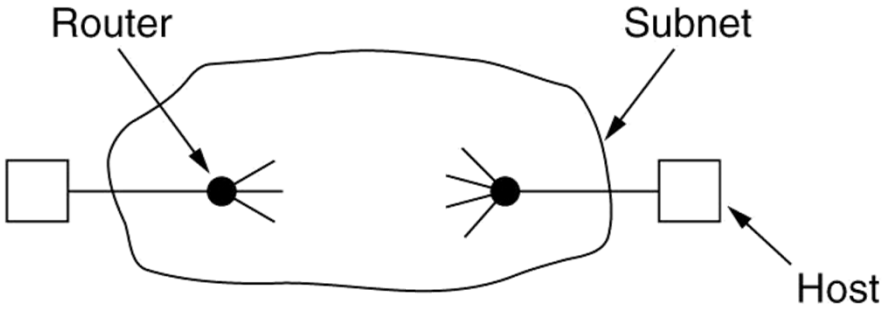
Curtin University

- Flow control in **TL** ~= Flow control in **DL**
  - ✓ But not the same
  - ✓ Both use sliding window or other scheme

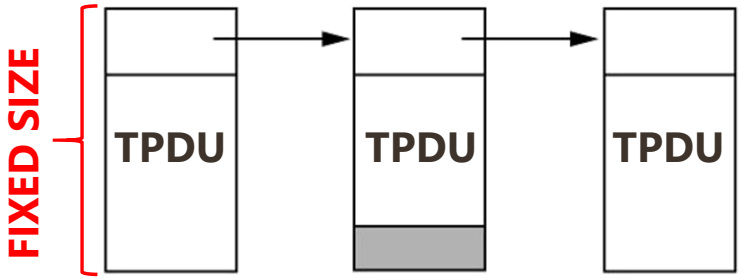


- A router usually has relatively few connection lines

- A host may have numerous connections
  - ✓ transport entity must buffer all TPDUs sent, same reason for data link layer
  - ✓ data link layer buffering strategy cannot be used



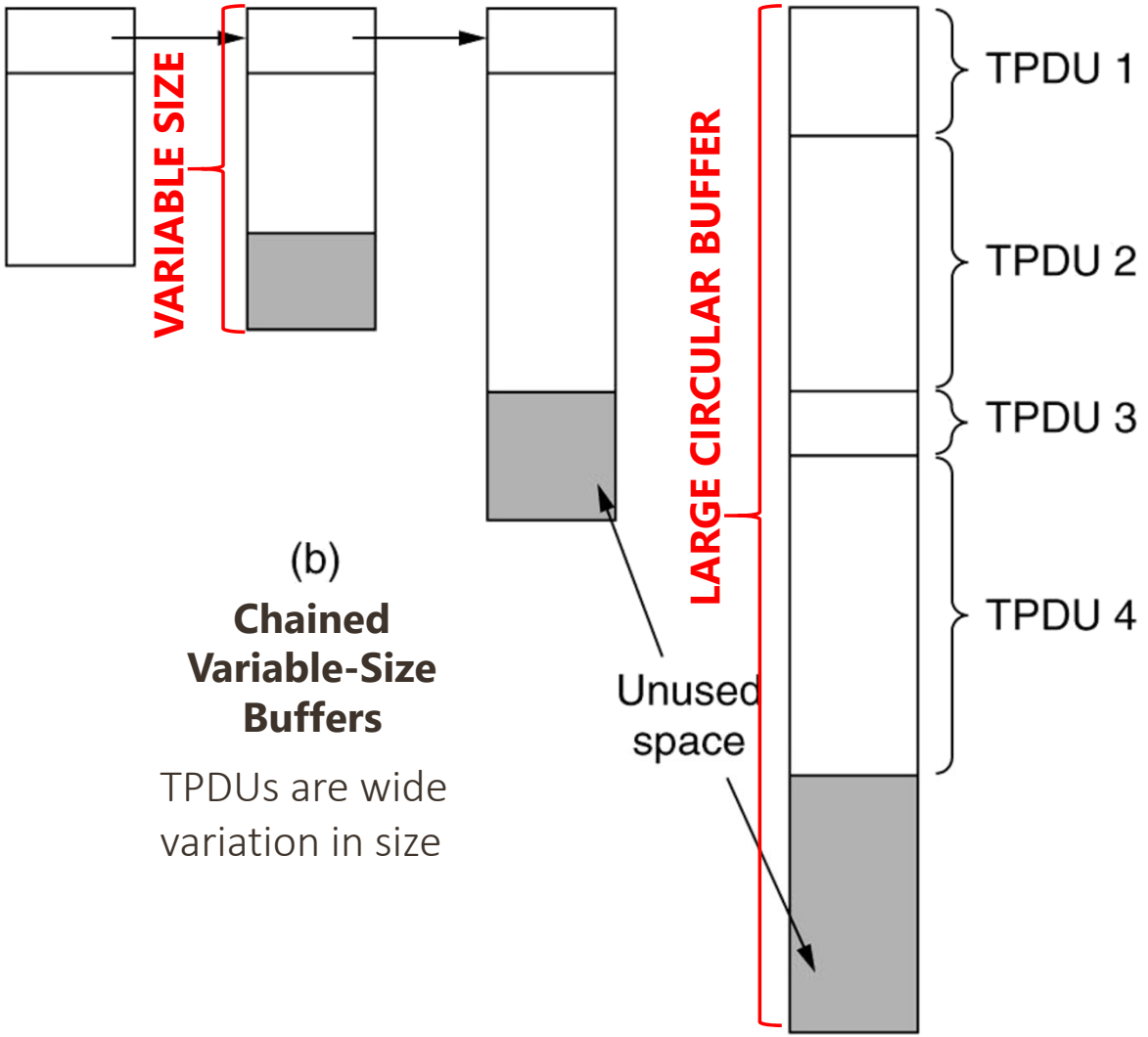
- Source Buffering & Destination Buffering



(a)

Chained Fixed-size Buffers

Most TPDUs are nearly the same in size



(b)

Chained Variable-Size Buffers

TPDUs are wide variation in size

Circular Buffer Per Connection (c)

BUFFERING METHODS

Source / Destination Buffering

Curtin University

- ✓ **low-bandwidth, bursty traffic**: *buffer at the sender*
- ✓ **high-bandwidth, smooth traffic**: *buffer at the receiver*

Dynamic Buffer Management

more on this later !

- ✓ decouple the buffering from the acknowledgement
- ✓ a **variable-sized window**

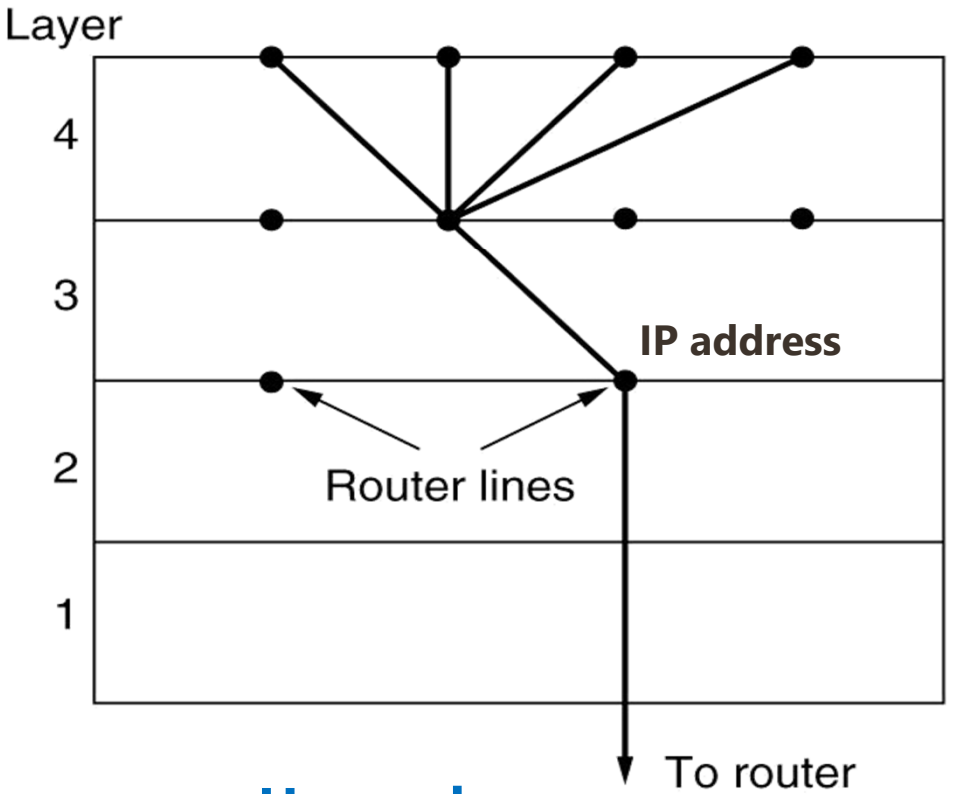
As memory price fall – equip hosts with more memory.
BUT, the **carrying capacity of the subnet is the bottleneck**

?

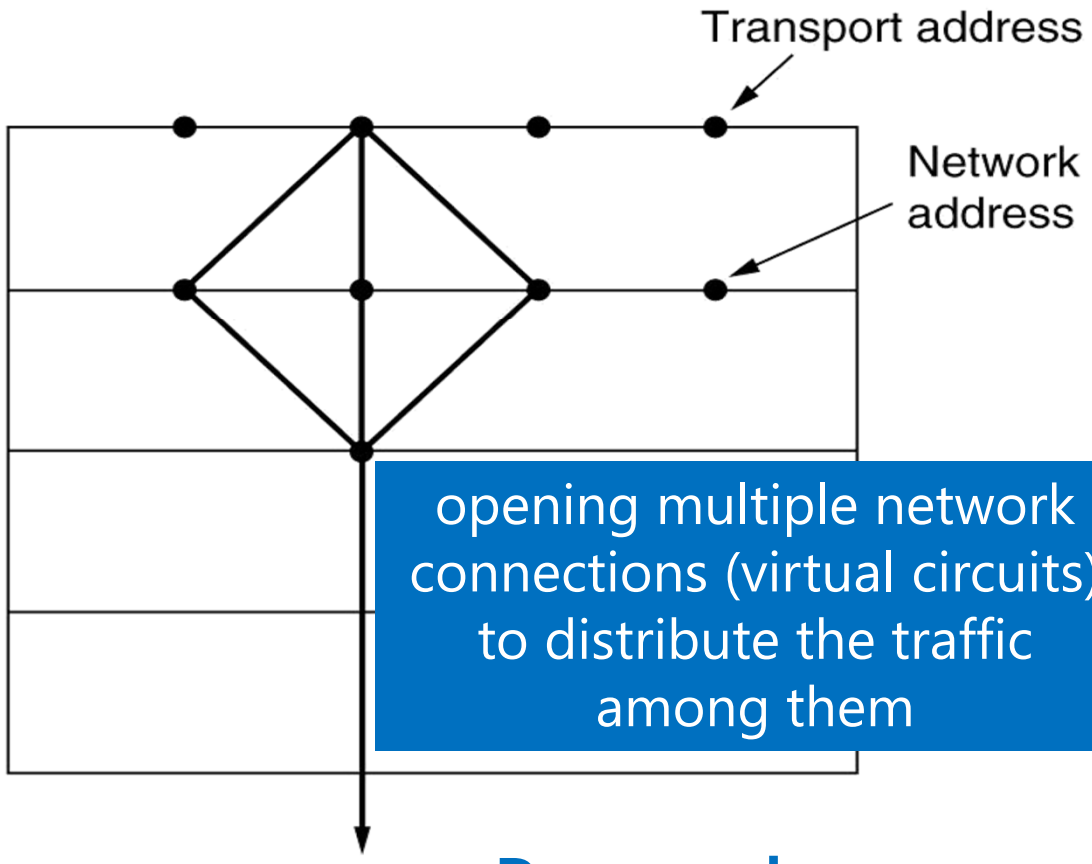
solution: mechanism based on subnet's carrying capacity rather than on the receiver's buffering capacity.

5. Multiplexing

Curtin University



Upward Multiplexing (a)



opening multiple network connections (virtual circuits) to distribute the traffic among them

(b) **Downward Multiplexing**

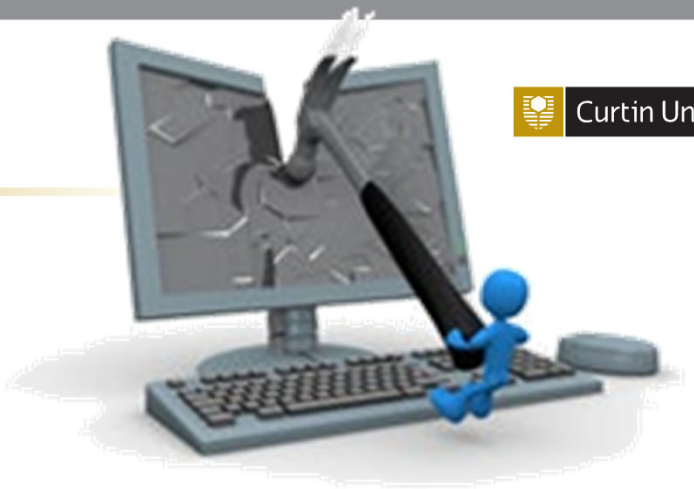
Crash Recovery

Curtin University

1. Router / Network Crash

- **For datagram service network:**
  - ✓ Transport entities expect lost TPDUs all the time and know how to cope
- **For connection-oriented network:**
  - ✓ Loss of virtual circuit is handled by establishing a new one and probing the remote transport entity for TPDUs received and not received

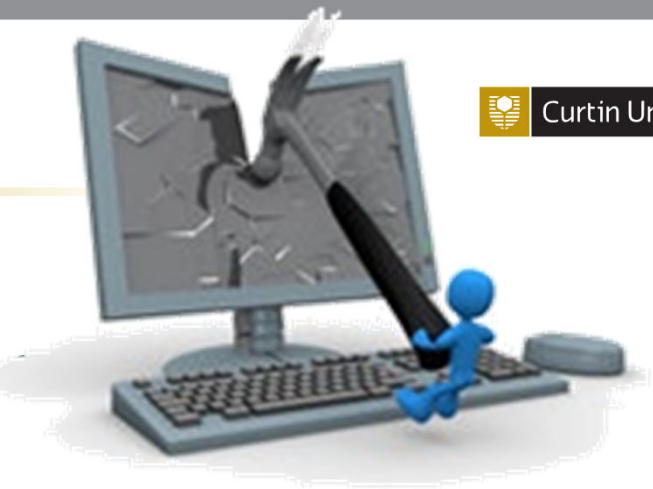
Crash Recovery



2. Host Crash

- After reboot tables/parameters are reinitialized
- To recover previous status:
  - ✓ Server (receiver) might broadcast TPDU to all other hosts, announcing that it had just rebooted and request that its clients (senders) inform it of the status of all open connections. Each client (sender) can be in one of the two states
    a. **No TPDU**s outstanding, **S0**
    b. **One TPDU** outstanding, **S1**
- Server (receiver) can be programmed in one of two ways:
  - ✓ **Send ack**nowledgement **First** (A)
  - ✓ **Write to app**lication process **First** (W)

Crash Recovery



Curtin University

2. Host Crash – cont.

- Client (sender) can be programmed in one of the
  - ✓ **always** retransmit the last TPDU
  - ✓ **never** retransmit the last TPDU
  - ✓ retransmit only in **state S0** (No TPDU outstanding)
  - ✓ retransmit only in **state S1** (TPDU outstanding)

There are **always** situations where the protocol **fails to recover** properly

Curtin University

FALIURES

Strategy used by receiving host

Strategy used by sending host	First ACK, then write			First write, then ACK		
	AC(W)	AWC	C(AW)	C(WA)	W AC	WC(A)
Always retransmit	OK	DUP	OK	OK	DUP	DUP
Never retransmit	LOST	OK	LOST	LOST	OK	OK
Retransmit in S0	OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1	LOST	OK	OK	OK	OK	DUP

OK = Protocol functions correctly
DUP = Protocol generates a duplicate message
LOST = Protocol loses a message

Curtin University

Transport Control Protocol (TCP)

- Fundamentals
- TCP Header
  - Flags (SYN, FIN, ACK, RST)
  - Flag (URG, PSH) – in depth
  - TCP Options
  - Window Size (Dynamic Buffer Management)
- TCP Flow Control
  - Dynamic Buffer Management

TCP

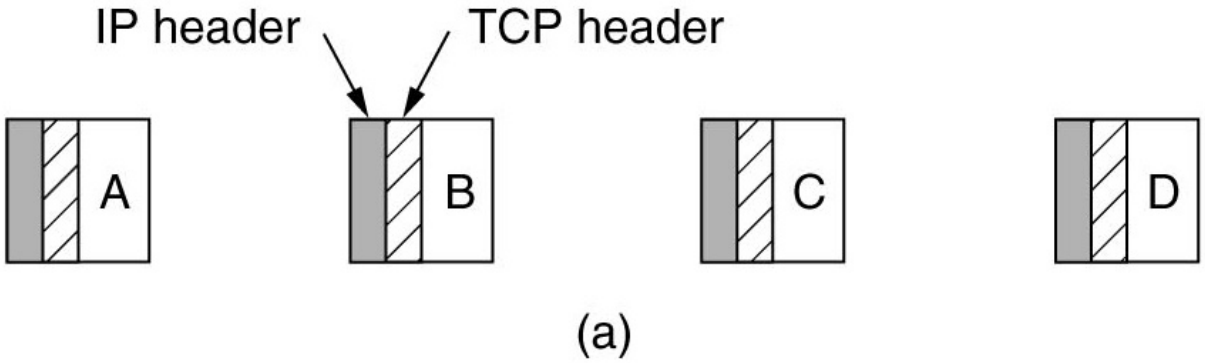
Curtin University

- The (other) **main** transport **protocol** used in the **Internet**
  - ✓ **Connection-oriented** protocol
  - ✓ RFC 793 (formal), RFC 1122 & 1323 (bug fixes)
  - ✓ Provide a reliable end-to-end communication over an unreliable internetwork
- **Connections** are:
  - ✓ **Full duplex** and point-to-point
  - ✓ **A byte stream** not a message stream

No support for Multicasting or Broadcasting !

TCP: Header

Curtin University

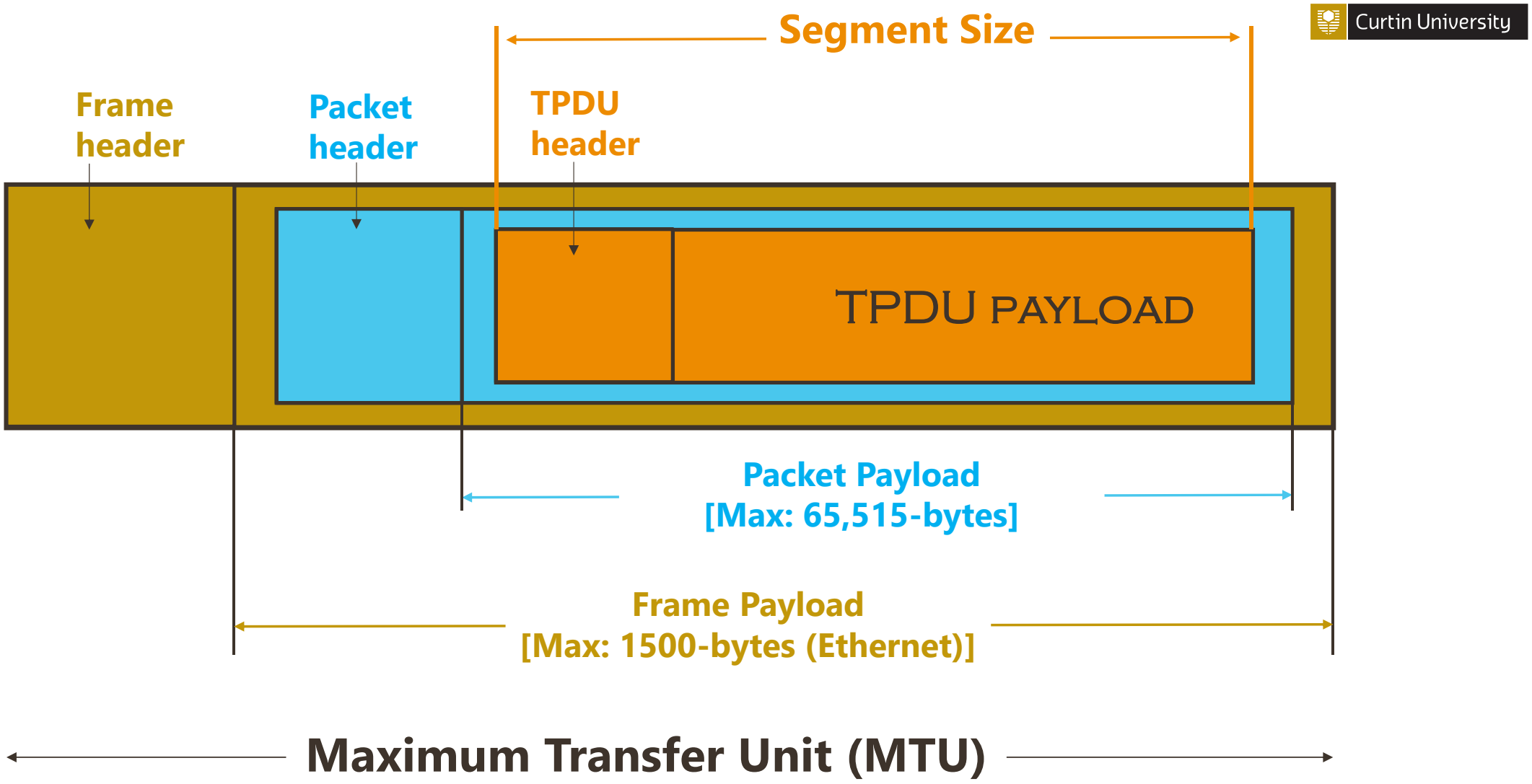


(a)

512-byte segments
sent as a separated IP datagrams

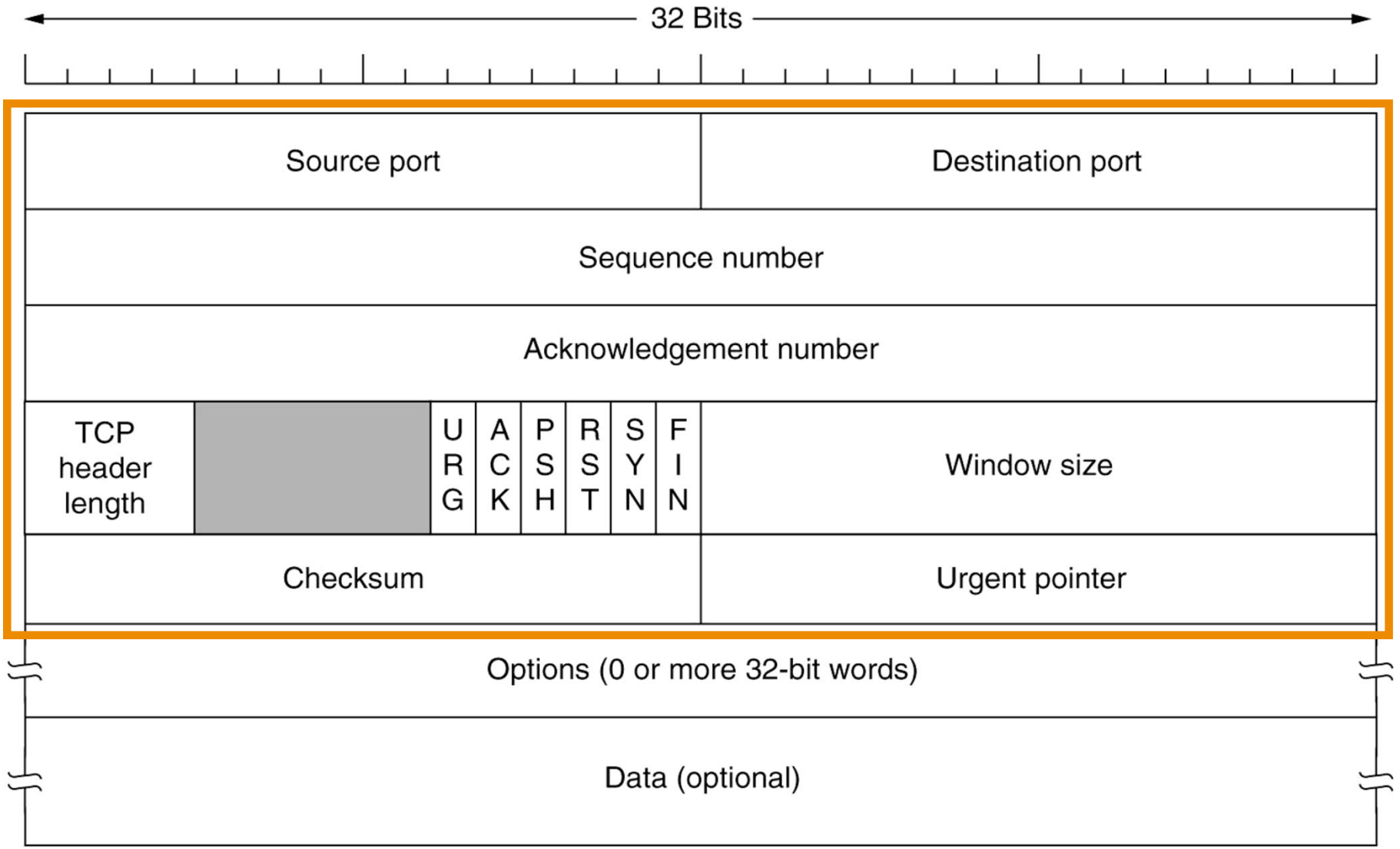
(b)

2048-bytes of data
received to the application in a single **READ** call



Curtin University

TCP HEADER

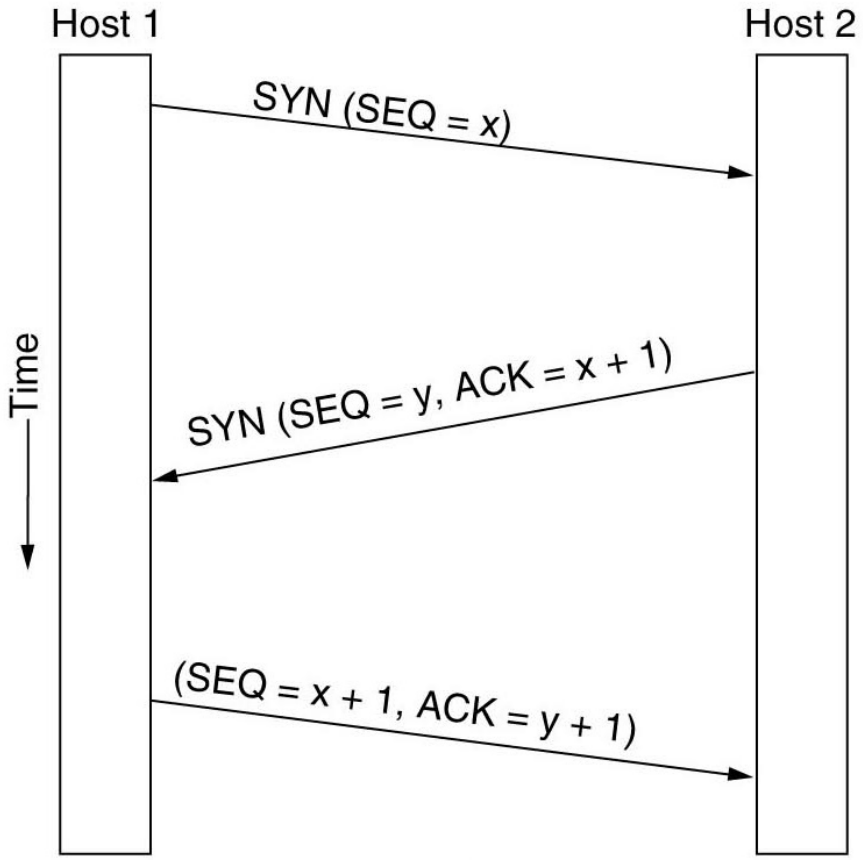


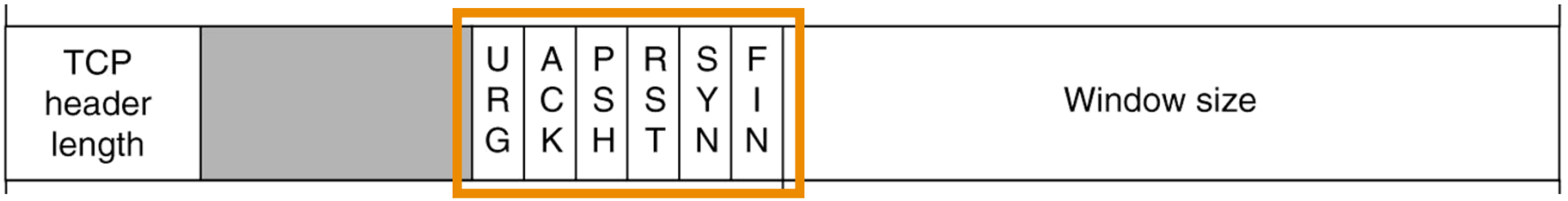
Fixed 20-bytes header

TCP: Header – cont.

Curtin University

- **Sequence Number (32-bit)**
  - sequence number of the first data octet (byte) in this segment
  - if SYN is set this field is the initial sequence number (ISN) and the first data octet is ISN+1
- **Acknowledgement Number (32-bit)**
  - sequence number of the next data octet the TCP entity expects to receive. May be piggybacked!!
  - NOTE that TCP is byte or stream oriented.

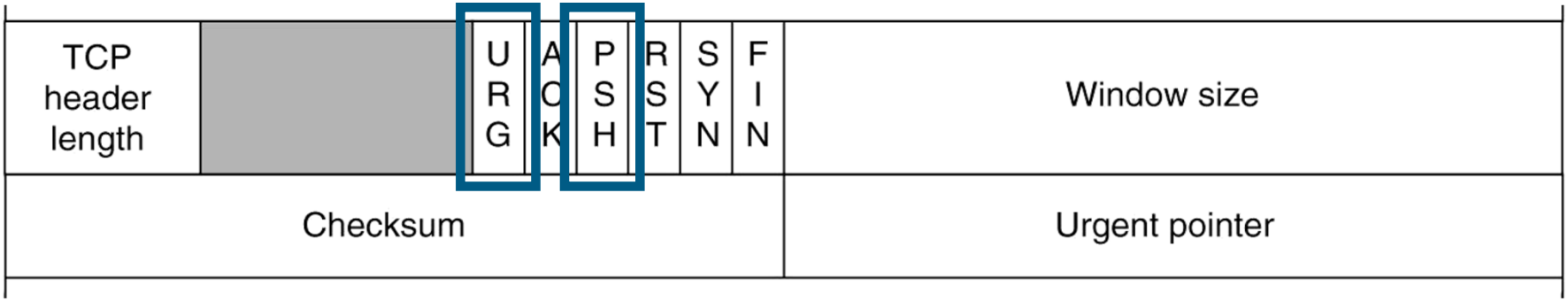




Curtin University

FLAGS

- ✓ **URG:** Urgent pointer field significant. Inform the destination TCP user that 'urgent' data is arriving.
- ✓ **ACK:** Acknowledgement field significant.
- ✓ **PSH:** Push function. A TCP user can require TCP to send (receive) all outstanding data up to and including that labelled with a **PUSH** flag.
- ✓ **RST:** Reset the connection.
- ✓ **SYN:** Synchronize the sequence numbers. Used to establish connections.
- ✓ **FIN:** No more data from sender. Used to release connections.



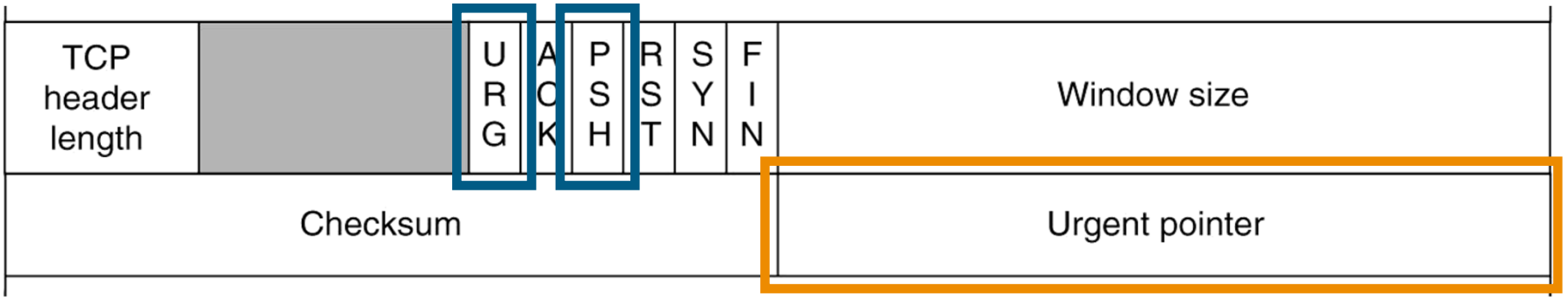
Curtin University

FLAGS

*When application passes data to TCP, TCP may **send** it **immediately** or **buffer it (at both sides)***
(in order to collect a larger amount to send at once)

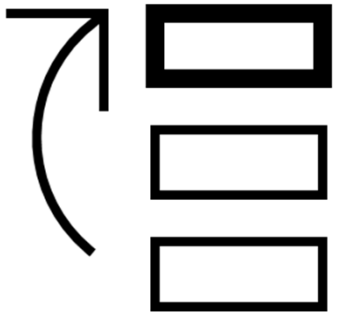
- **PUSH Flag – (used by application) ->** Force/Flush data out  ***at both sides!***
- **URGENT Flag – (used by application) ->** Cause TCP to stop accumulating data and transmit everything it has for the connection immediately

when urgent data is received, receiving application is **interrupted** (stops whatever it was doing) & read the data stream to find the urgent data



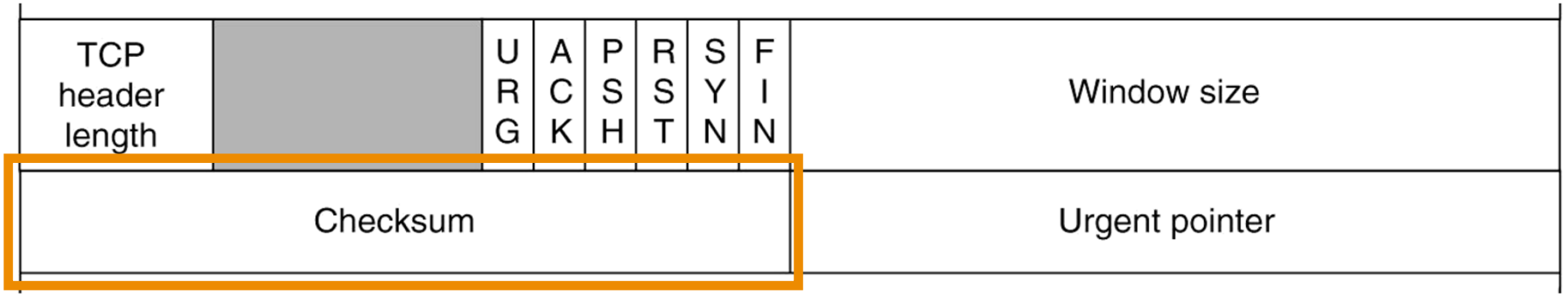
Curtin University

URG POINTER



Urgent Pointer

- ✓ **Points to the last octet** in a sequence of urgent data. Allows the receiver to know how much urgent data is coming

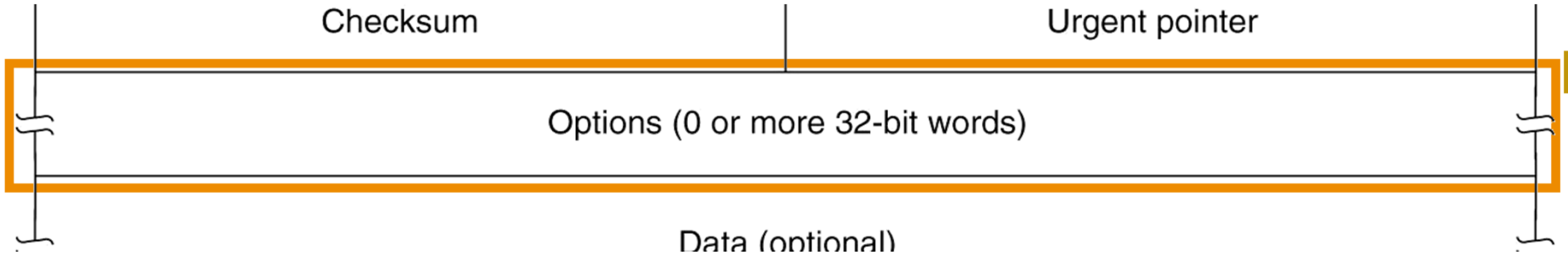


Curtin University

- **Checksum:**
  - ✓ **header**
  - ✓ **data**
  - ✓ conceptual **pseudo-header**

CHECKSUM

source & destination addresses, segment length. This provides protection from mis-delivery



Curtin University

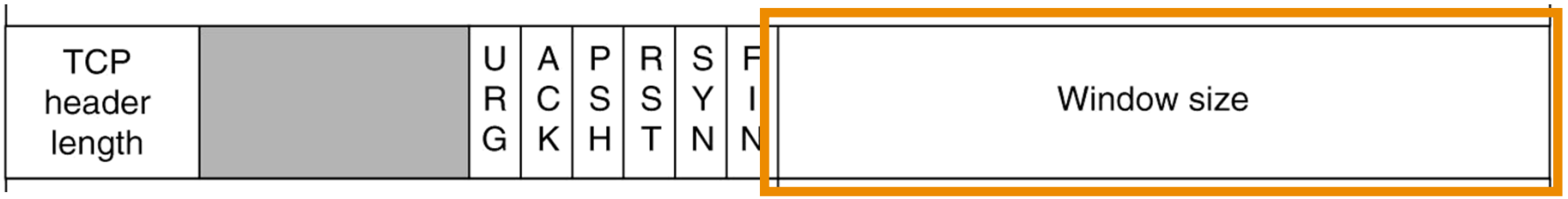
OPTIONS

- ✓ **Maximum Segment Size (MSS)**
- ✓ **Window Scale Factor:**

  Window is multiplied by 2F where F is the window scale factor.

- ✓ **Timestamp**

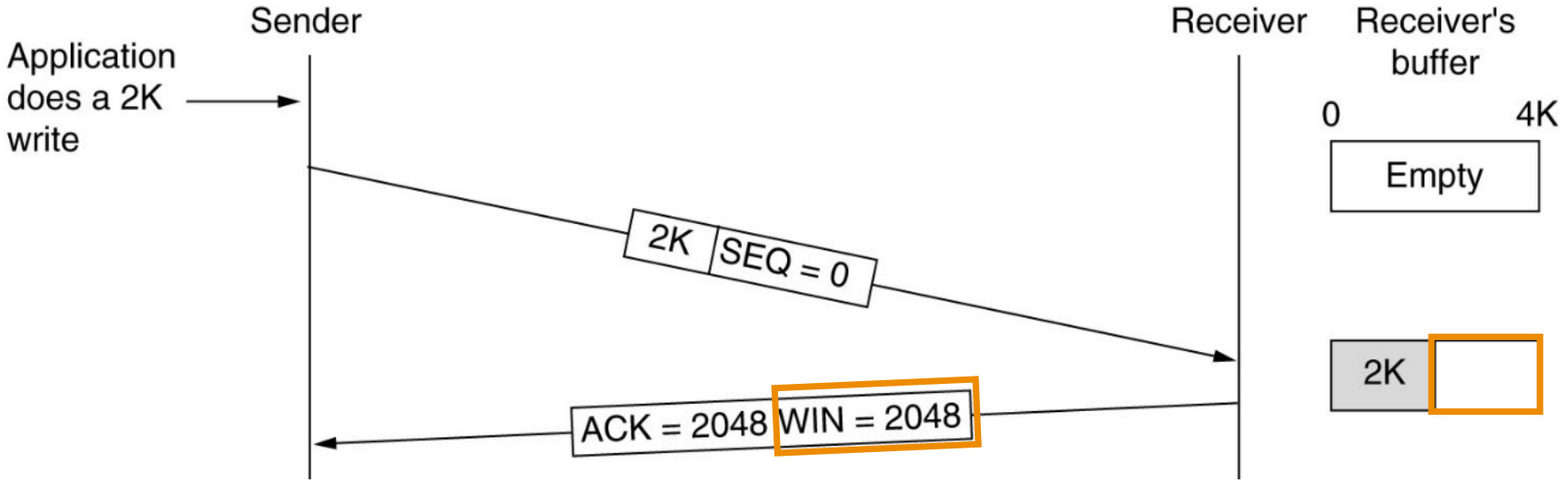
  Any outgoing packet with a timestamp will cause the ACK to carry a timestamp echo with the same value. Can be used to calculate round trip time



Curtin University

- ✓ Used in flow control
- ✓ **Variable-sized** Sliding Window

WND SIZE



Window Size

Curtin University

- **Window Management:** not tied to ACKs *(differs from data link protocols)*
- Receive entity will **advertise** window segment that it can receive & buffer
- Each entity can **alter size** of the other's sending window **dynamically** using the segment's Window field.

DYNAMIC BUFFER MANAGMENT

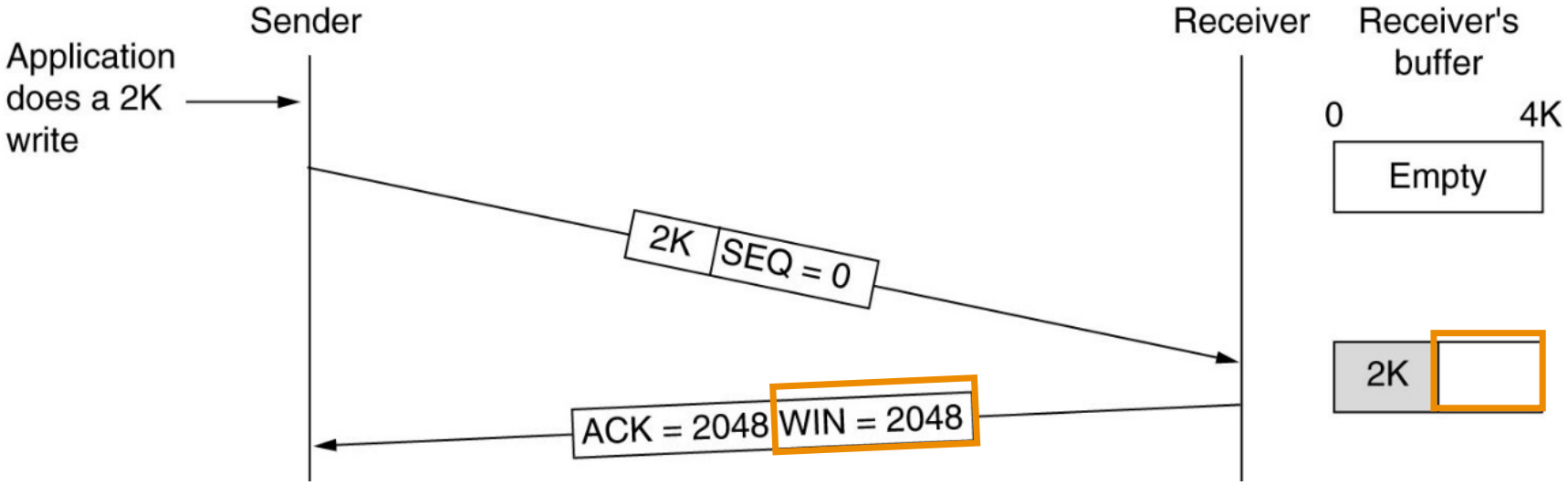
TCP: Flow Control

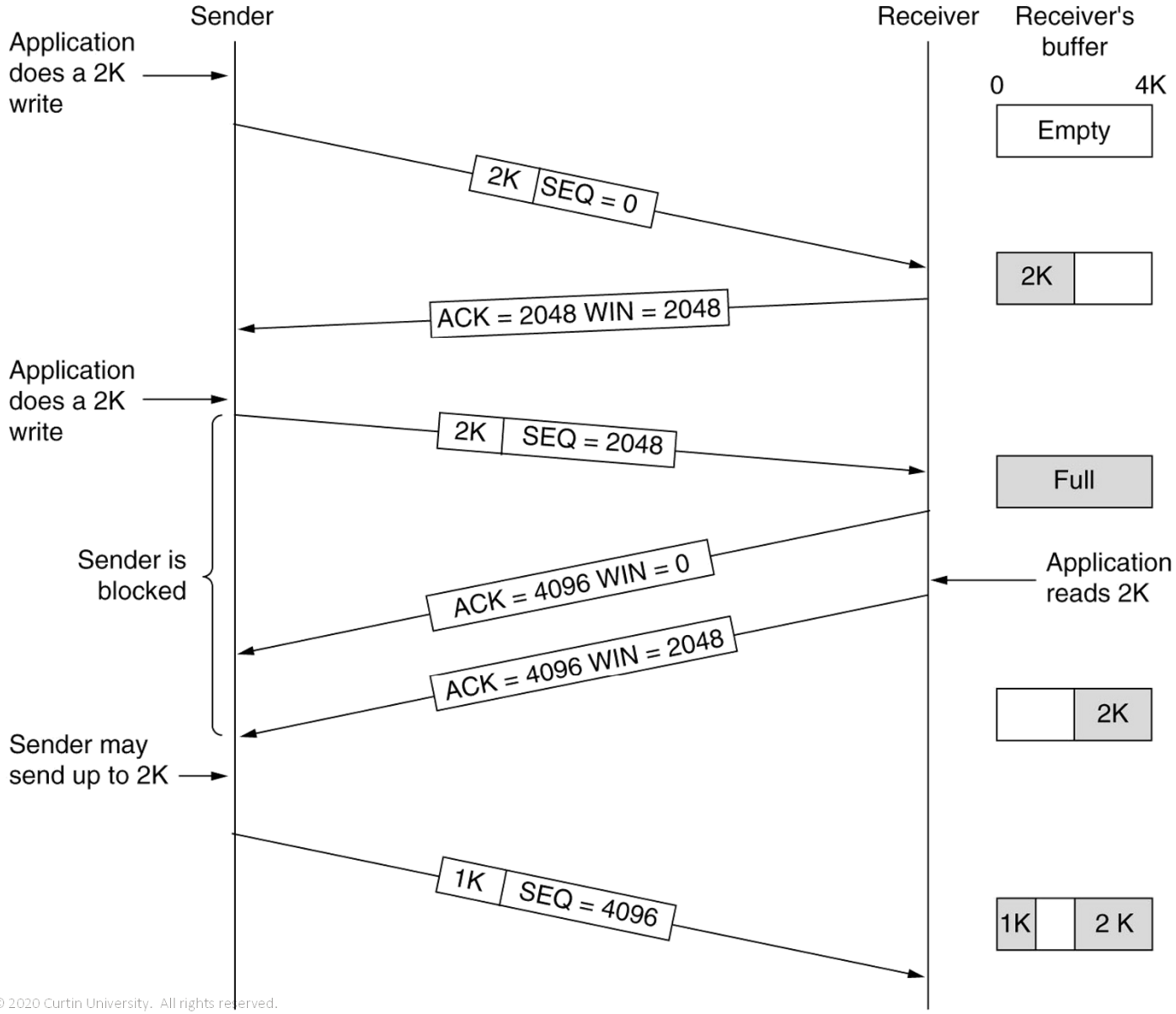
Curtin University

Each entity implement flow control using a **credit mechanism**, also called a **window advertisement**.

A **credit specifies** the maximum number of bytes the entity sending this segment can receive and buffer from the other entity

DYNAMIC BUFFER MANAGMENT

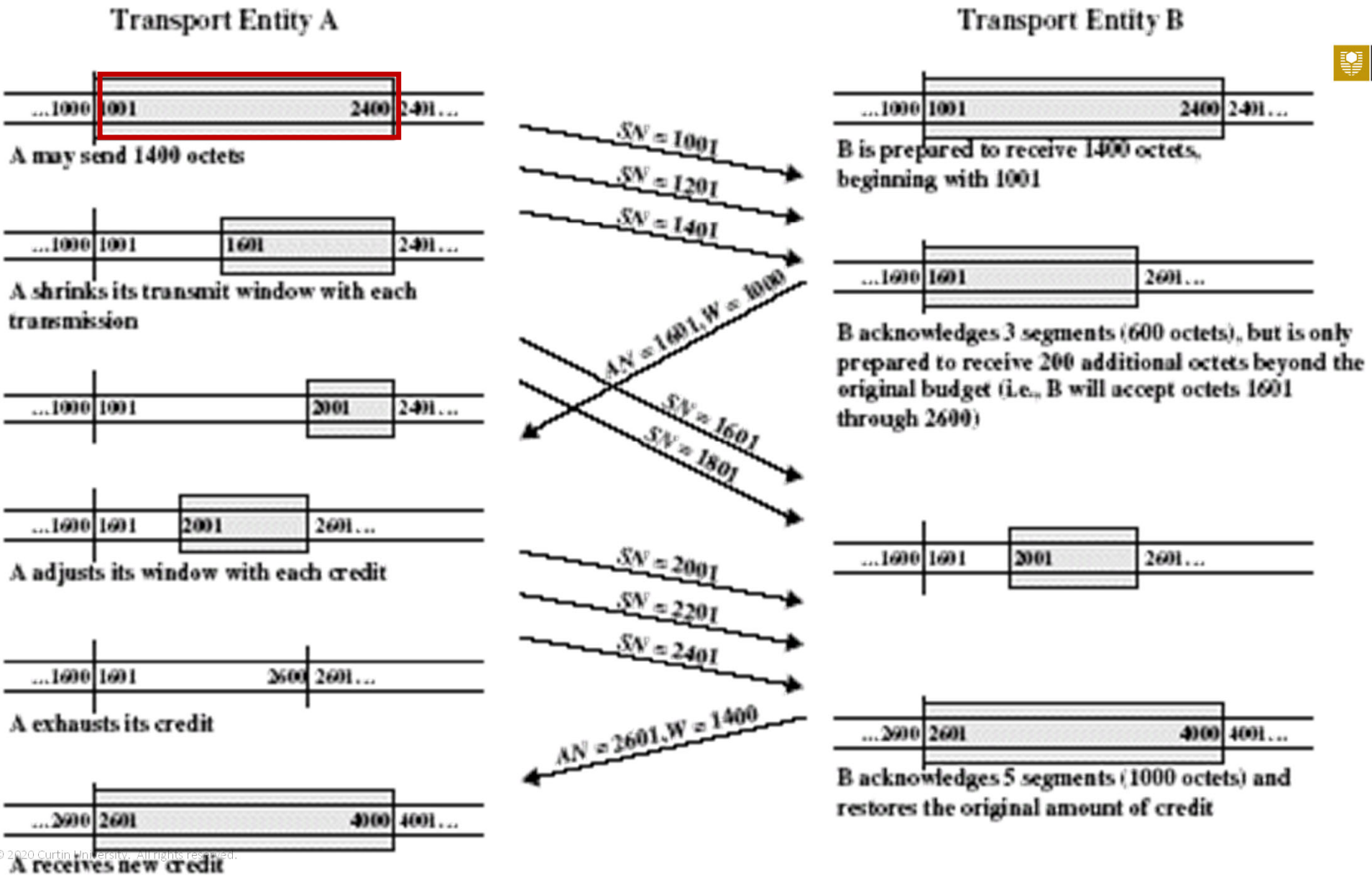




Curtin University

DYNAMIC BUFFER MANAGMENT

TCP flow control the **sequence number refers to byte sequence** instead of packet (or segment) sequences.



Curtin University

TCP Implementation Policies

- Send Policy
  - Silly Window Syndrome
- Deliver Policy
- Accept Policy
- Retransmit Policy
- Acknowledge Policy

TCP: Implementation Policies

Curtin University

- The standard provide a precise specification of the protocol to be used between TCP entities

- Policies defined for the protocol
  1. **Send policy**
  2. **Deliver policy**
  3. **Accept policy**
  4. **Retransmit policy**
  5. **Acknowledge policy**

1. Send Policy

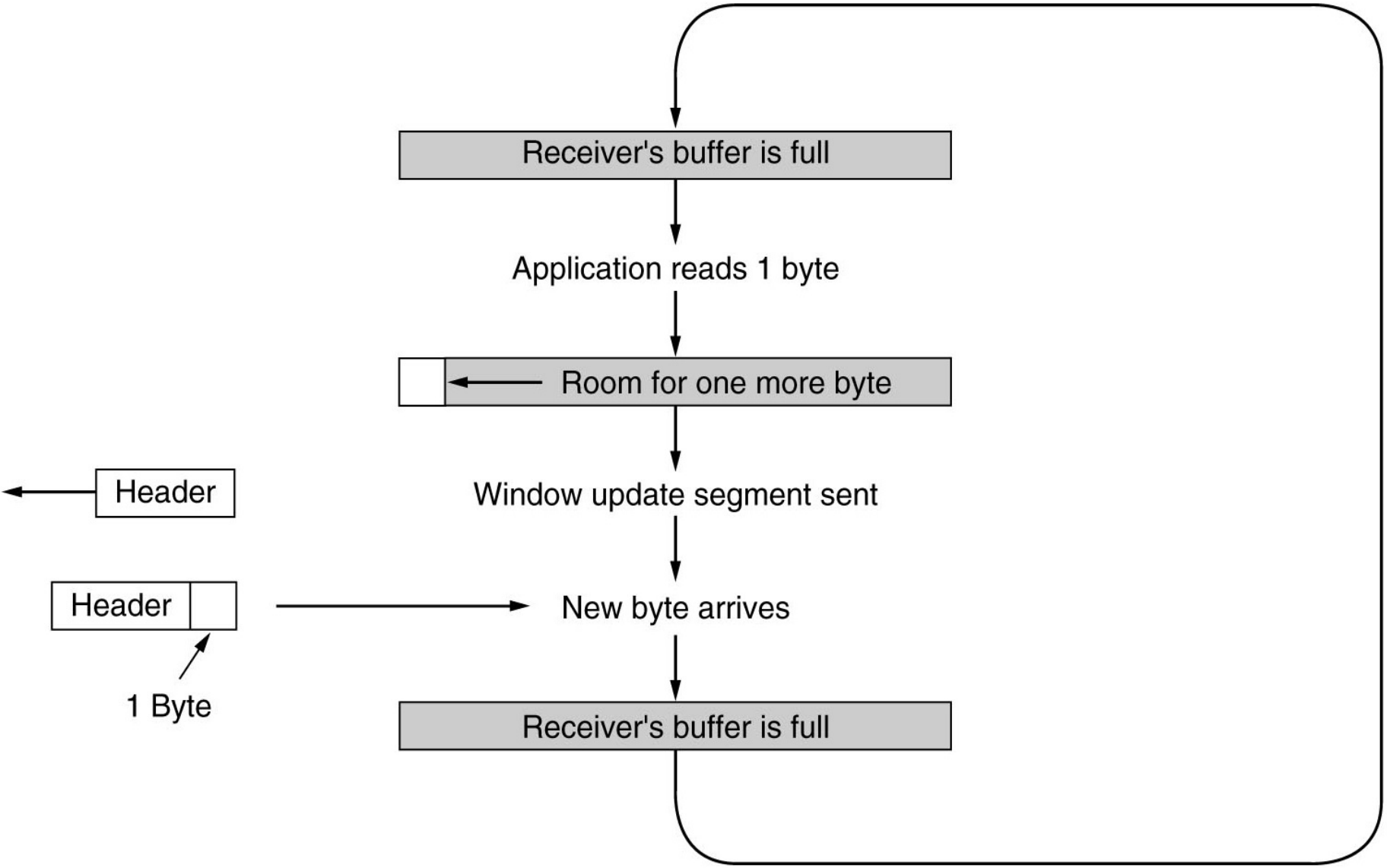
Curtin University

In the **absence of PUSH** data and a closed transmission window, a sending TCP entity is free to **transmit data at its own convenience**

- depend on performance considerations
  - ✓ if **infrequent and large** *(buffer @ sender)*
    - low overheads
    - slow response
  - ✓ if **frequent and small** *(buffer @ receiver)*
    - quick response
    - high overheads
    - silly window syndrome

Curtin University

Silly Window Syndrome



2. Deliver Policy

Curtin University

Too frequent delivery means **too many OS interrupts**

- Arriving data are stored in deliver buffer

✓ **if PUSH flag is set**

Data along with any other data in the deliver buffer are submitted to the destination application in a *RECEIVE* command.

✓ **if PUSH flag is not set**

TCP may wait, *e.g. to avoid excess interrupts*

✓ **if URG flag is set**

The receiving application is signaled that urgent data is present

3. Accept Policy

Curtin University

- **In Order**
  - ✓ **Discard** out of order segments

- **In Window**
  - ✓ **Accept** all segments within the receive window
  - ✓ Complex acceptance test and sophisticated storage

4. Retransmit Policy

Curtin University

TCP maintains a **queue of segments** that have been sent but **not ACKed**

- **First Only** suited for **in-window accept policy**
  - ✓ one retransmission timer for the entire queue
  - ✓ if an ACK is received, the segment/s removed and timer reset
  - ✓ if timer expires, first segment in the queue is retransmitted

Selective-Repeat ARQ

- **Batch** suited for **in-order accept policy**
  - ✓ same as above, except when timer expires, retransmit all segments in the queue

Go-Back-N ARQ

- **Individual** suited for **in-window accept policy**
  - ✓ one timer for each segment

Selective-Repeat ARQ

5. Acknowledge Policy

Curtin University

- **Immediate**
- **Cumulative**
  - ✓ wait for an outbound segment, piggyback the ACK
  - ✓ Timer to avoid long delay

Curtin University

- **Transport Layer**
  - Fundamentals
  - Transport Entity
- **Transport Layer Elements**
  - Addressing
  - Connection Establishment
  - Connection Release
  - Flow Control and Buffering
  - Multiplexing
  - Crash Recovery
- **TCP**
  - TCP Header
    - Flags (SYN, FIN, ACK, RST)
    - Flag (URG, PSH) – *in depth*
    - TCP Options
    - Window Size (Dynamic Buffer Management)
  - TCP Flow Control
    - Dynamic Buffer Management
- **TCP - Implementation Policies**
  - Send Policy
    - Silly Window Syndrome
  - Deliver Policy
  - Accept Policy
  - Retransmit Policy
  - Acknowledge Policy


Curtin University
314
THANK YOU
Make tomorrow better.