NSCOM01

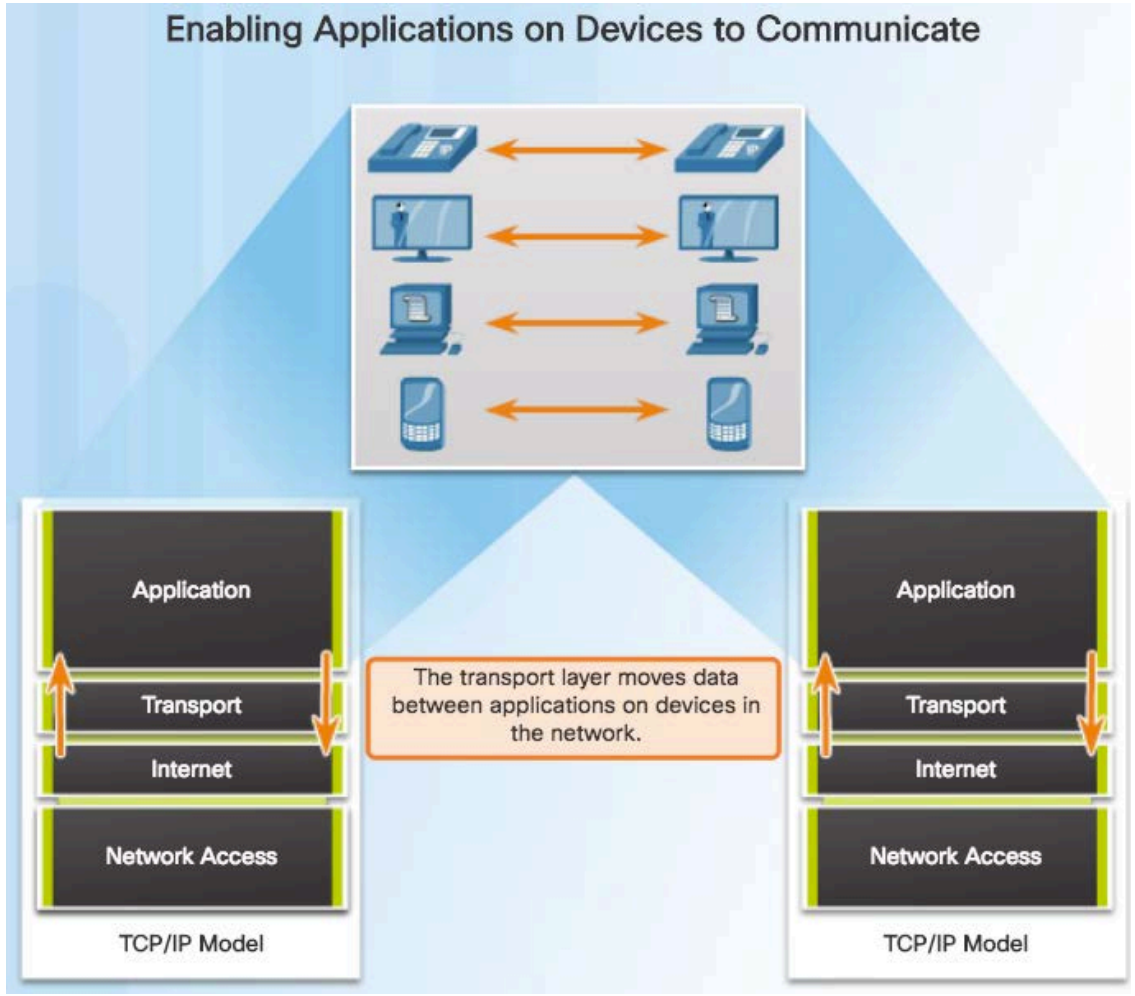
Unreliable Data Delivery

3rd Term AY 2022-2023

Instructor: Dr. Marnel Peradilla

SPIRAL REVIEW: TRANSPORT LAYER

- **Responsible for establishing a temporary communication session between two applications and delivering data between them.**
- **Link between the application layer and the lower layers that are responsible for network transmission.**



SPIRAL REVIEW: TRANSPORT LAYER

❑ Tracking the Conversation

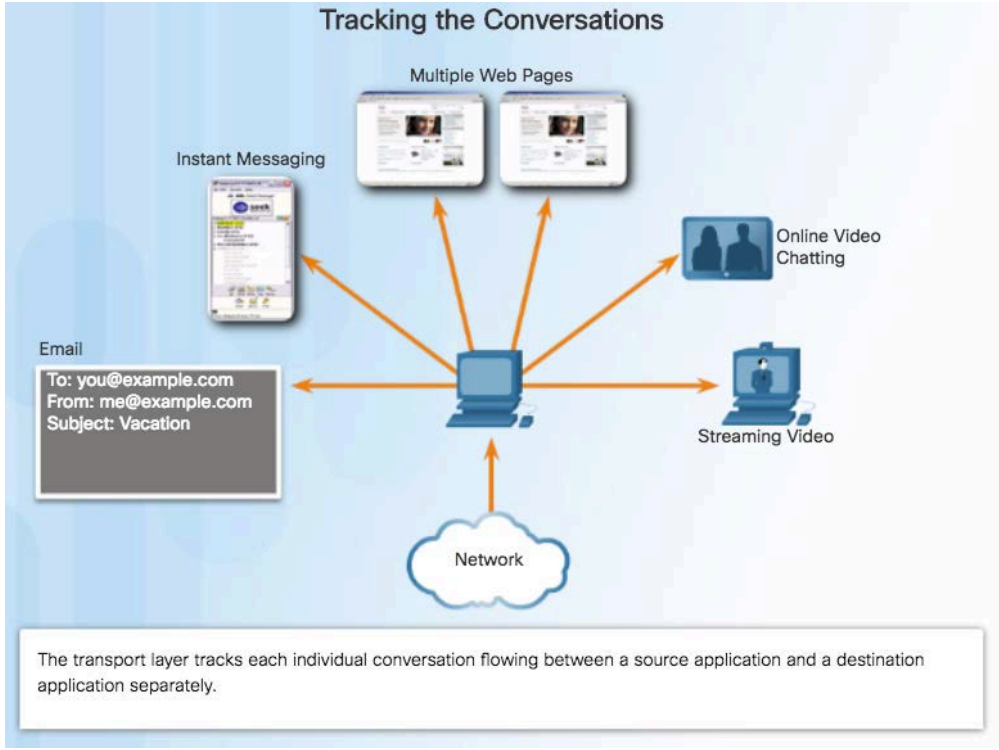
- Tracks each individual conversation flowing between a source and a destination application.

❑ Segmentation

- Divides the data into segments that are easier to manage and transport. Header used for reassembly is used for tracking.

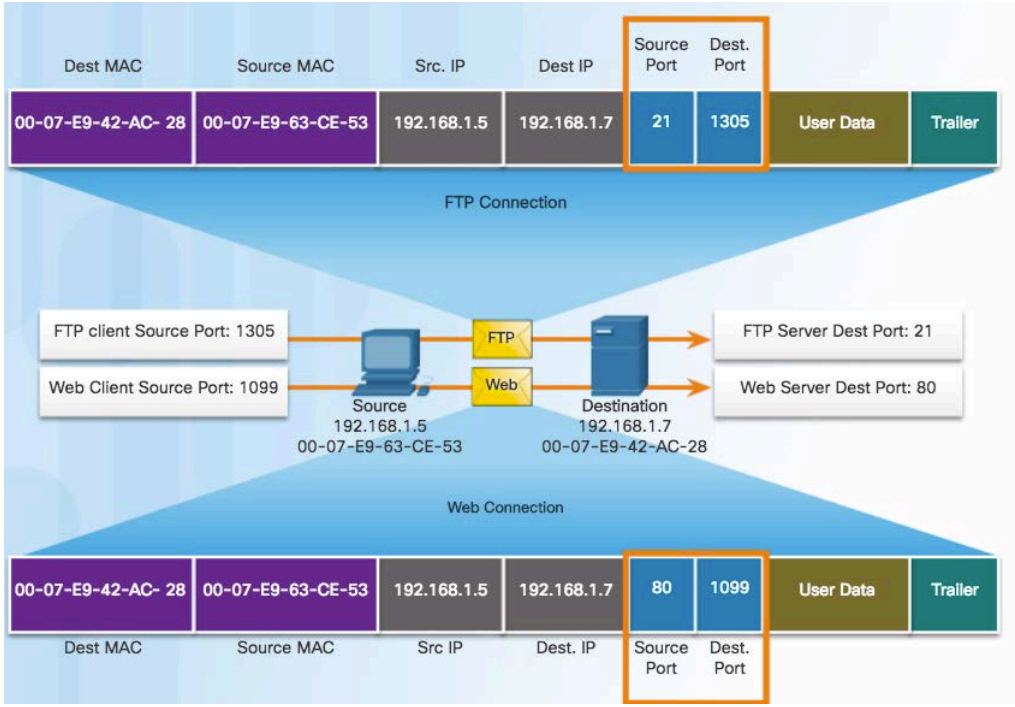
❑ Identifying the Application

- Ensures that even with multiple applications running on a device, all applications receive the correct data via port numbers.



SPIRAL REVIEW: TRANSPORT LAYER

- **The transport layer of the TCP/IP protocol suite uses ports to differentiate data transfers occurring simultaneously on the same host**
  - Well-known Ports (0 to 1023) - reserved for services and applications.
  - Registered Ports (1024 to 49151) - assigned by IANA to a requesting entity to use with specific processes or applications.
  - Dynamic Ports (49152 to 65535) - assigned dynamically by the client's OS and used to identify the client application during communication.



CONNECTIONLESS SERVICES

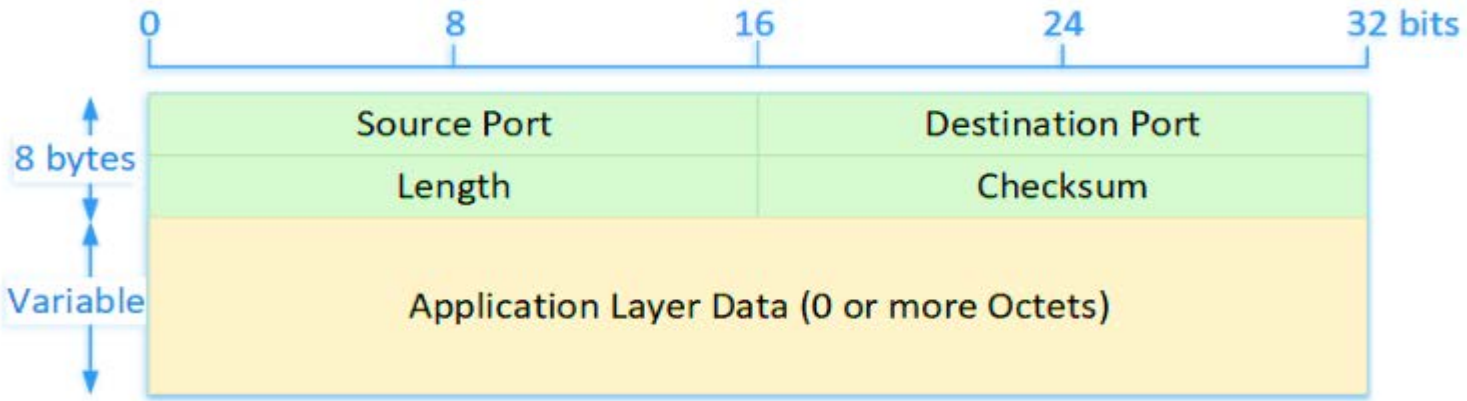
- **Commonly used with applications where occasional data loss is tolerable in exchange for reduced protocol overhead:**

1. Inward Data Collection – periodic sampling of data sources such as sensors or automatic self-test reports from network equipment
2. Outward Data Dissemination – message broadcasting to nodes or distribution of data to a network
3. Request – Response – query-based applications that use a transaction service provided by a single server where a single request-response is typical
4. Real-time applications – applications with a degree of redundancy or real-time requirement e.g. voice, telemetry

USER DATAGRAM PROTOCOL

- **The User Datagram Protocol (UDP) is a connectionless transport protocol used in TCP/IP networks**
- **Considered as a 'bare-bones' protocol that provides only the essential capabilities needed to transport a data segment between applications**
- **Features:**
  1. Unreliable – datagrams are not acknowledged
  2. No congestion control mechanism- datagrams sent as quickly as possible
  3. Stateless – Server does not keep track of status and session information of a client. Each request-response exchange with a client is treated as an independent transaction
  4. Unordered delivery – datagrams do not contain any sequencing information

UDP HEADER



- Source Port (Optional): indicates the port of the sending process and is assumed to be the port where a reply is to be sent. Set to 0 if unused
- Destination Port: port number of receiving process
- Length: length in octets of the UDP segment datagram including header and the data
- Checksum (Optional): Used for error detection, calculated based on packet header, transport header and payload bits. Set to all 0's if not used

UDP-BASED APPLICATION PROTOCOLS

- **Several well-known application protocols use UDP as transport protocol to support their operations:**
  - System Logging Protocol (syslog) – Message logging protocol used to convey event notification messages from network applications or devices
  - Network Time Protocol (NTP) - Used to synchronize the time of a computer client or server to another server or reference time source
  - Domain Name System (DNS)
  - Dynamic Host Configuration Protocol (DHCP)
  - Trivial File Transfer Protocol (TFTP) - a lightweight file transfer mechanism used for transferring short configuration files to routers and other devices, typically over a short dedicated link or within a LAN environment
  - Simple Network Management Protocol (SNMP) - Protocol for collecting and modifying information about managed devices on IP networks

CONSIDERATION WHEN USING UDP

- **When using a best-effort transport protocol like UDP, an application may need to implement mechanisms to address the following issues:**
  - Connection Establishment and Termination
  - Data Error Detection and Recovery
  - Ordered Delivery
  - Retransmission Strategy
  - Duplicate Detection
  - Crash Recovery
  - Congestion Control

CONSIDERATION WHEN USING UDP

❑ Connection Establishment and Termination

- Applications using unreliable transport protocols may require mechanisms to signal the start and end of data transfer:
  - Transferring a stream of data that is segmented into multiple datagrams
- To create some semblance of a connection, the following may be done:
  - Application level commands that initiate a transaction with a host
  - Application commands that signify the end of a transaction
- Consequently, the following are possible using UDP because no connection is needed:
  - Using a broadcast to simultaneously transmit to clients instead of sending multiple copies of the same segment
  - Have a single socket accepting traffic from multiple sources

CONSIDERATION WHEN USING UDP

❑ Ordered Delivery

- Maximum segment lifetime (MSL) of UDP is 2 minutes. Application needs to be able to handle any segment that arrives within this time interval
- Segments may be complete but arrive out of order
- If application needs to reconstruct a data stream from several datagrams:
  - Needs to incorporate numbers for sequencing
  - Needs the capability to reassemble the original data stream from its individual datagrams

CONSIDERATION WHEN USING UDP

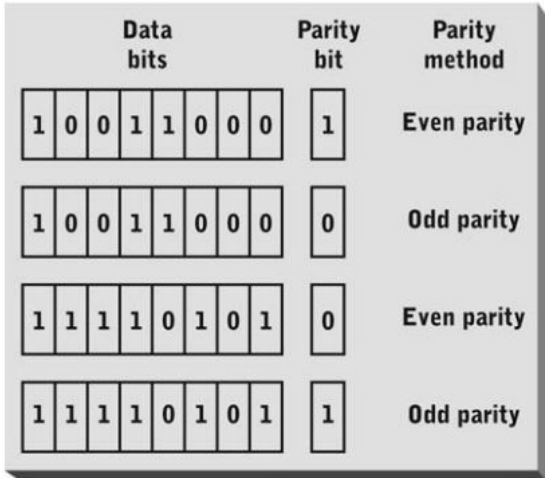
❑ Data Error Detection and Correction

- As with any data transmission, UDP segments may be damaged in transit due to issues such as noise on transmission lines or physical layer damage
- UDP header checksum uses a weak error detection algorithm
- Application using UDP that require data integrity such as those transmitting high value data or those that support critical systems may implement additional error checking mechanisms to handle data corruption.
- Type of error checking used is dependent on the probability of errors on the connection and the importance of data accuracy of the application
- Checksums can be used if only error detection is needed
  - Sender computes checksum based on data then transmits checksum together with data
  - Receiver computes checksum based on received data
  - Receiver compares computer checksum and received checksum and accepts data only if checksums are equal

CONSIDERATION WHEN USING UDP

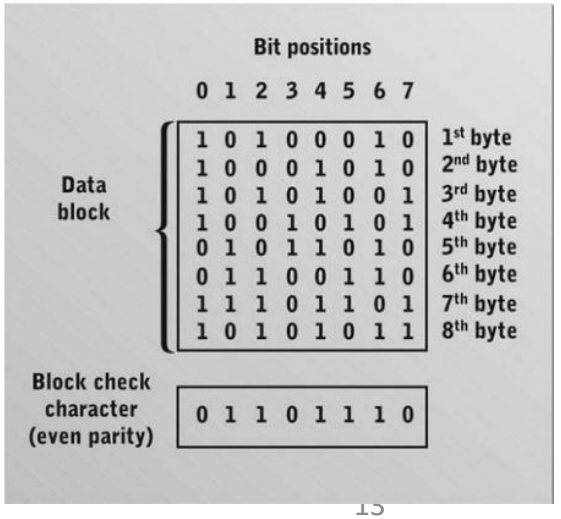
❑ Data Error Detection and Correction

- Parity Checking a.k.a. vertical redundancy checking
  - Count number of 1's in the data.
  - Even parity; - set parity bit to '0' of count of 1 bits is even, '1' if odd
  - Odd parity: set parity bit to '0' of count of 1 bits is odd, '1' if even.



- Block Checking a.k.a. Longitudinal redundancy checking
  - Sending device counts the number of 1-valued bits at each bit position in a block, then combines parity bits for each position into a block check character (BCC) and adds it to the end of the block
  - Vulnerable to the same types of errors as parity checking

- Cyclic Redundancy Check (CRC)
  - Most widely used error-detection method.
  - Produces a BCC for a group of characters or bytes but uses more than 8 bits and a different mathematical algorithm



CONSIDERATION WHEN USING UDP

❑ Data Error Detection and Correction

- Forward Error Correction (FEC) Codes may be used if errors need to be corrected
  - Receiver may correct errors on its own provided that errors are limited to a few bits only
  - Avoids the additional cost of having to request for a retransmission
  - Requires additional redundant bits to be sent
  - Triple module redundancy code example
    - ‘**0**’ sent as ‘**000**’ and ‘**1**’ sent as ‘**111**’
    - A value ‘**0101**’ is sent as ‘**000111000111**’
    - A received value ‘**000101000111**’ is detected as an error and automatically corrected to ‘**000111000111**’

CONSIDERATION WHEN USING UDP

❑ Retransmission Strategy

- Retransmission of data may be needed when:
  - Segment arrives at the destination but is damaged in transit
  - Segment does not arrive at the destination
- An application that needs confirmation of successful transmission of data may implement the following:
  - Acknowledgments for received segments
  - Retransmission timer to limit waiting time for a segment to be acknowledged before retransmission
  - Application-level commands that signify a request for a missing or damaged segment

CONSIDERATION WHEN USING UDP

❑ Duplicate Detection

- Duplicate transmissions can happen under the following scenarios:
  - The acknowledgment for a segment is lost in transit prompting the sender to retransmit a copy
  - A transmitted segment is delayed hence the intended recipient does not acknowledge before the sender's timer expires and retransmits a copy
- An application that needs the ability to handle and differentiate potential duplicate segments must implement the following:
  - The receiver must assume that the acknowledgment was lost and acknowledge the duplicate
  - The segment sender must be able to handle multiple acknowledgments for the same segment
  - If implementing a sequence or identifying number for each segment, ensure that the range of values is large enough to not cycle within the expected lifetime of a segment
  - Use application commands that are repeatable without side effects

CONSIDERATION WHEN USING UDP

❑ Crash Recovery

- When the system upon which an application is running fails, any ongoing data transfer is halted, and the other side of the communication may not realize the failure that occurred.
- An application where a single stream of data may require multiple segments to be transferred with confirmation of receipt needs to handle receiver crashes so as not to enter an indefinite waiting state
- Handling methods:
  - Implementation of maximum retransmission attempts in case the receiving side does not recover
  - Implementation of application-level commands to restart or resume the transfer in case the receiving side recovers immediately

CONSIDERATION WHEN USING UDP

❑ Congestion Control

- UDP does not have any flow control mechanisms; hence an application may potentially transmit segments at a rate that consumes all available bandwidth
- Applications that use UDP should incorporate mechanisms that control their transmission rate to avoid causing congestion
- Typical approaches:
  - Low Data Volume applications (e.g. transaction-based apps)
  - Limit send rate to 1 packet per round trip time
  - Bulk Data applications (e.g. streaming apps)
  - Monitor packet loss and/or round trip time on the network and adjust such that the UDP packet flow using a windowing technique to increase or decrease flow
  - Maintain send rate but dynamically resize segment based on congestion

SOCKETS

- A socket is a structure in memory that represents the endpoint for communication
- A pair of sockets belonging to each communicating process are connected together, to create communication channels between the processes.
- Each socket is associated with the address of the process, which includes both its IP address (identifying the host computer) and the port number
- Socket primitives are essentially commands (or functions) that control the operation of a network socket

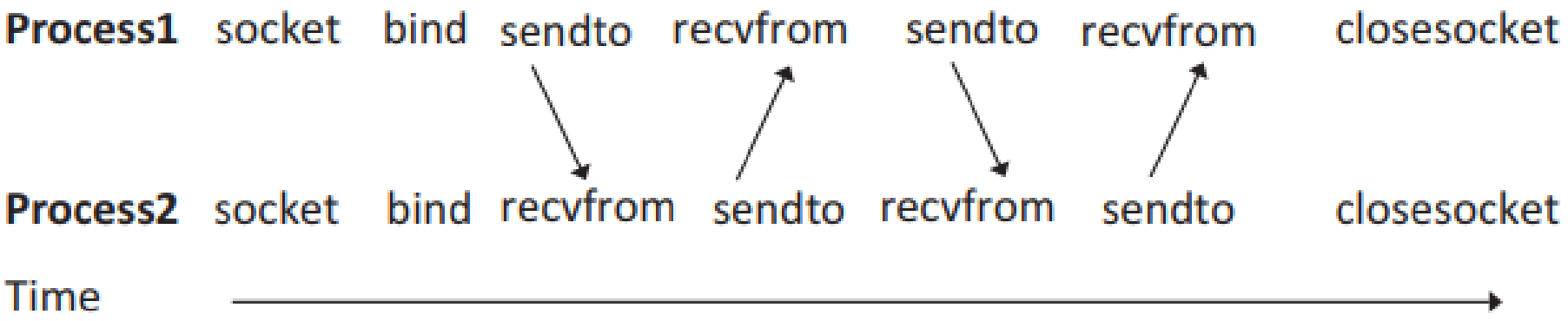
BINDING

- **Binding is the term referring to the association between a process and a port and is performed by the host OS**
- **Binding a process to a port requires that the port is not already in use by another process**
- **When a message arrives:**
  - The OS filters messages based on their destination IP address
  - The destination port number in the segment header is used to know which process to pass a particular arriving message to.
  - If there is no existing binding for the destination port, then the message is discarded

UDP SOCKETS

❑ UDP Sockets support the following primitives:

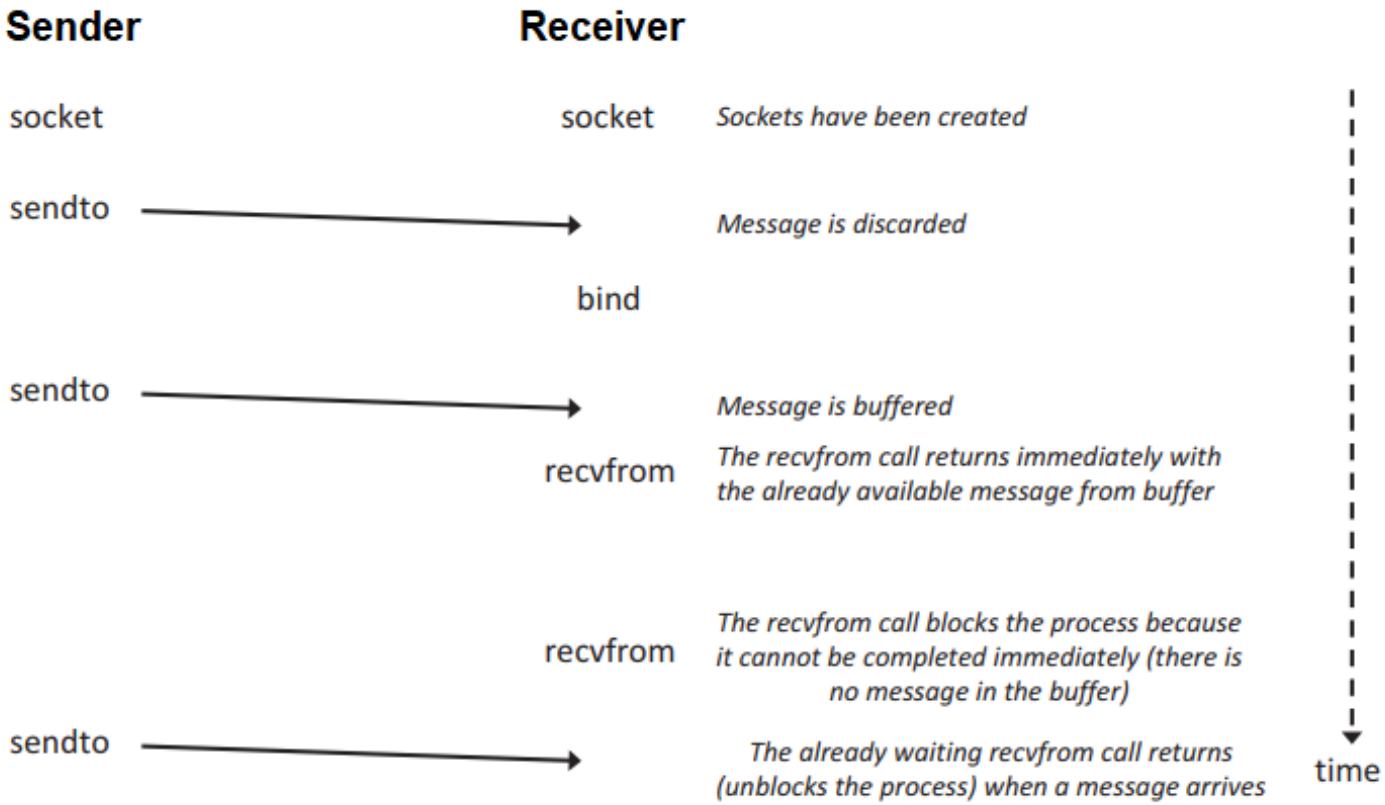
- Socket primitive is used to create a socket which may be set to blocking or non-blocking IO mode
- Bind primitive is used to map a process to a port
- Sendto primitive is used to send data to another process.
- Recvfrom primitive is used to retrieve data from the receive buffer.
- Closesocket primitive is used to close the socket.



UDP SOCKETS

❑ Blocking

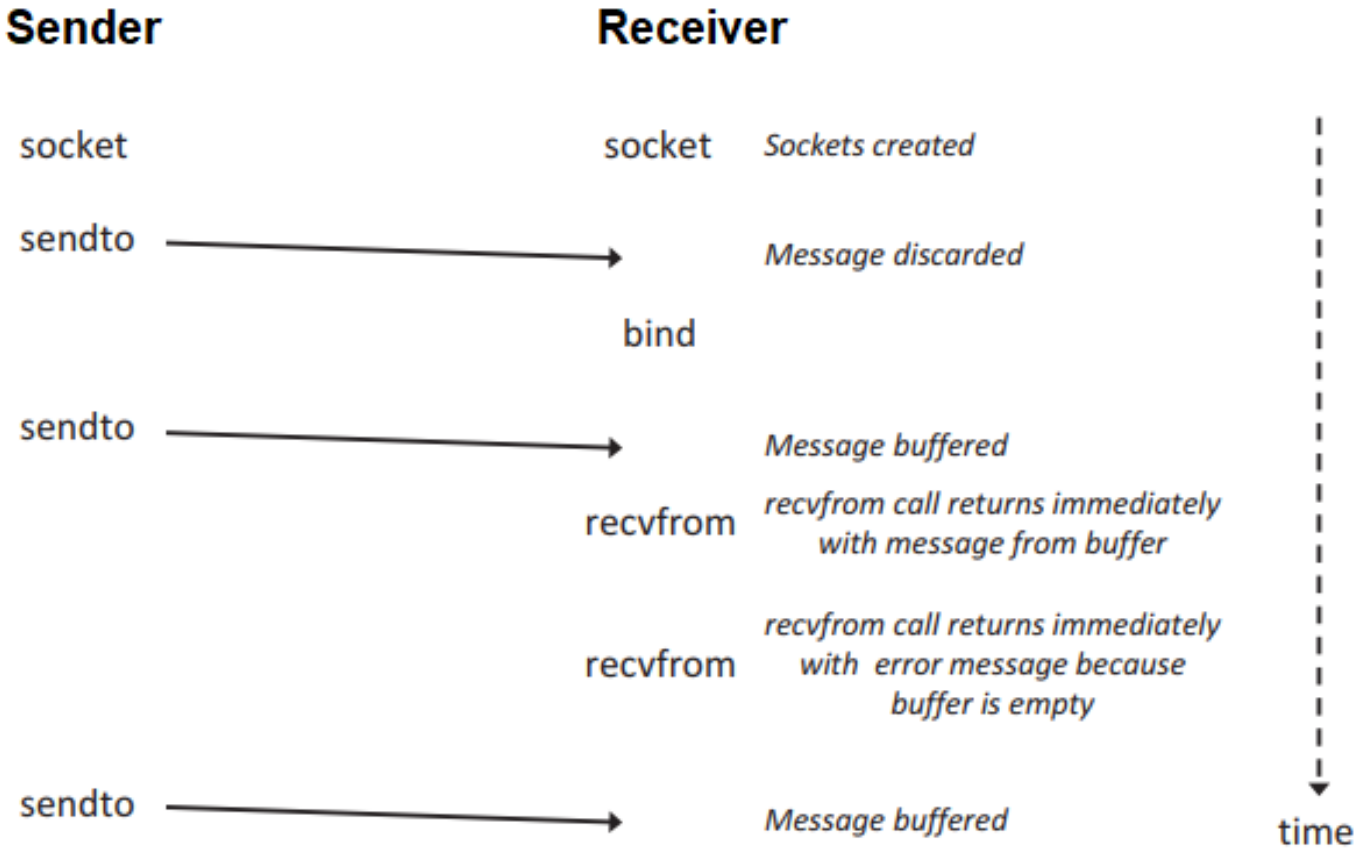
- Process has to wait for data to arrive(blocking state) when it attempts to receive and cannot do other activities while in this state



UDP SOCKETS

❑ Non-Blocking

- Process gets an error when it attempts to receive non-existing data and can choose to carry on with other activities then retry receiving later
- Working with non-blocking sockets usually entails the use of a timer to schedule periodic calls to the recvfrom primitive to check for new messages



RFC READINGS

- **UDP - RFC 768**

MESSAGE FROM DPO

"The information and data contained in the online learning modules, such as the content, audio/visual materials or artwork are considered the intellectual property of the author and shall be treated in accordance with the IP Policies of DLSU. They are considered confidential information and intended only for the person/s or entities to which they are addressed. They are not allowed to be disclosed, distributed, lifted, or in any way reproduced without the written consent of the author/owner of the intellectual property."