Institute for the Wireless Internet of Things
at Northeastern University

EECE5155: Wireless Sensor Networks and the Internet of Things

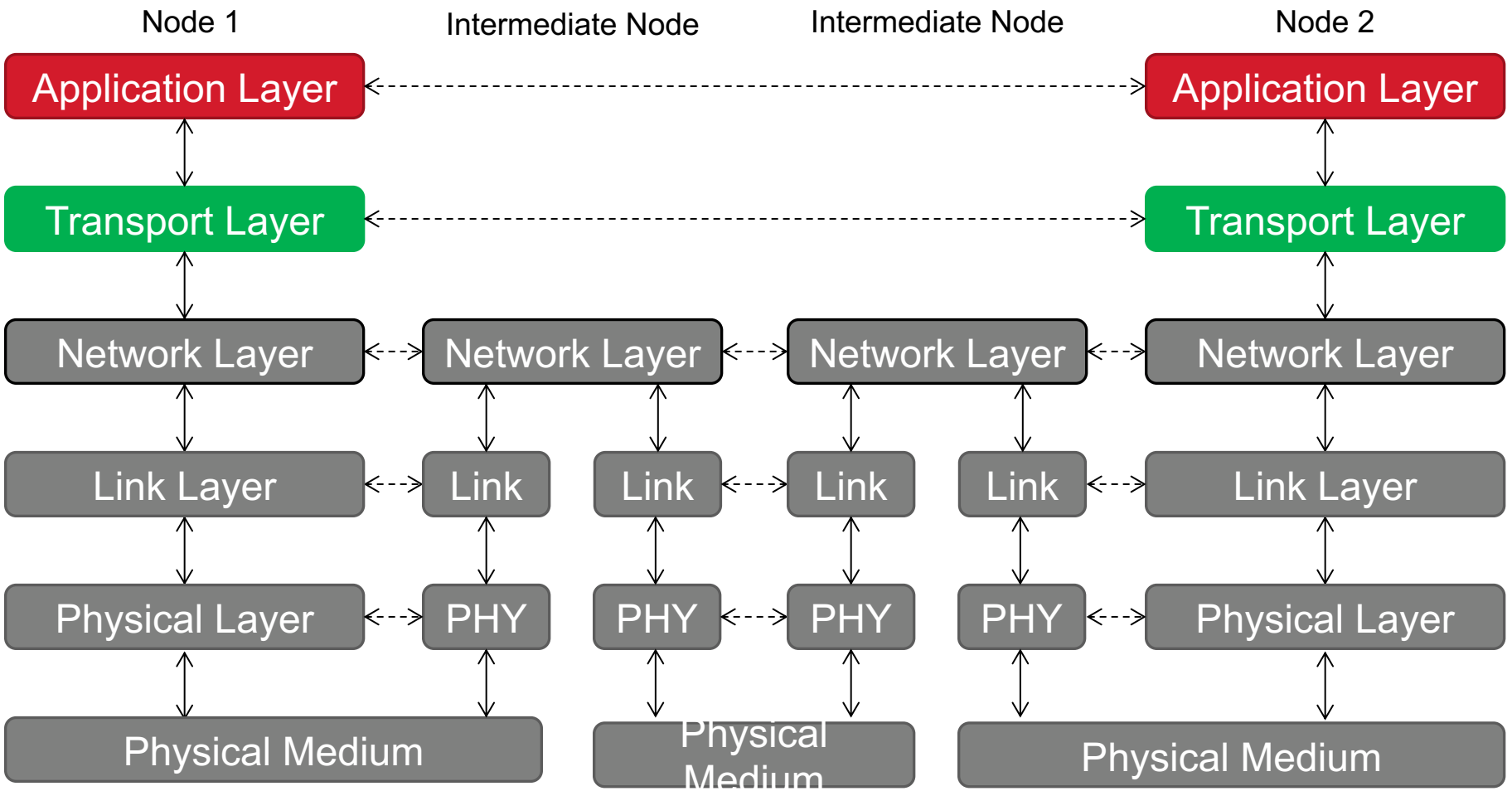
Josep Miquel Jornet
Associate Professor, Department of Electrical and Computer Engineering
Director, Ultrabroadband Nanonetworking Laboratory
Member, Institute for the Wireless Internet of Things
Northeastern University
jmjornet@northeastern.edu
www.unlab.tech

Institute for the Wireless Internet of Things at Northeastern

The Network Layer

The Protocol Stack

Institute for the Wireless Internet of Things at Northeastern



The Transport Layer

- **Objective:** To guarantee the correct end-to-end transmission of information over the network
  - This includes:
    - Reliable data transport
      - Deals with end-to-end packet losses
    - Flow Control
    - Congestion Control
      - Regulates transmission rates at the source to avoid losses due to congestion

Elements of Transport Protocols

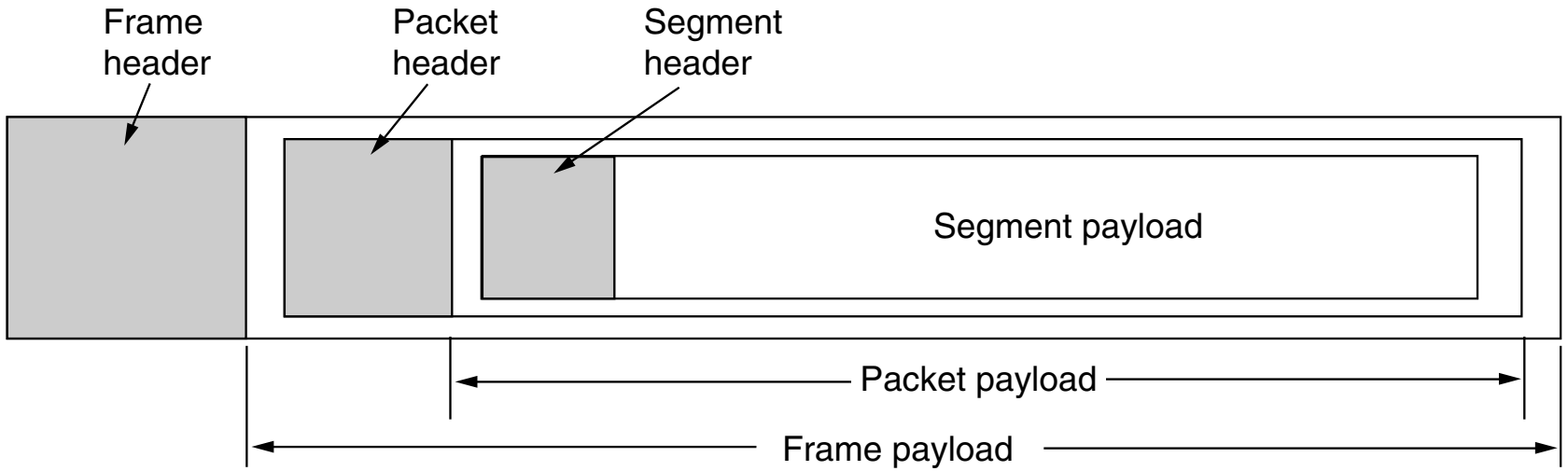
- The transport service is implemented by a transport protocol used between the two transport entities, normally at the two ends of the data path (e.g., **at the client and at the server**)

- In some ways, transport protocols resemble the link layer protocols:
  - Both have to deal with error control, sequencing and flow control, among other issues

- However, significant differences between the two also exist due to major dissimilarities between the environments in which the two protocols operate:
  - At the link layer, two routers communicate directly via a physical channel, whether wired or wireless
  - At the transport layer, this physical channel is replaced by the entire network

Segments

- Messages sent from transport entity to transport entity (compare to packets, frames)



Ports

- So far, we have talked about:
  - Link/MAC addresses at the link layer
    - Identify a node in a link
    - Are usually fixed/static (e.g., your WiFi card has a unique fixed MAC address)
  - Network/IP addresses at the network layer
    - Identify a node in a network
    - Are usually dynamic (e.g., your computer has a different IP address in different networks)
- New:
  - Ports are **addresses at the transport layer**
    - Identify a specific application executed in a node
    - Are generally different at the transmitter and the receiver

Ports

- For many common services, the port at which a server is listening is well-known:
  - **20: FTP (file transfer)**
  - **22: SSH, SFTP (secure file transfer)**
  - **25: SMTP (sending email)**
  - **80: HTTP (browsing)**
  - **110: POP 3 (retrieving email)**
  - **143: IMAP (email)**
  - **194: IRC (IRC chat)**
  - **443: HTTPS (secure HTTP)**
  - **…**

Common Transport Layer Protocols

- User Datagram Protocol (UDP)
- Transport Control Protocol (TCP)

User Datagram Protocol (UDP)

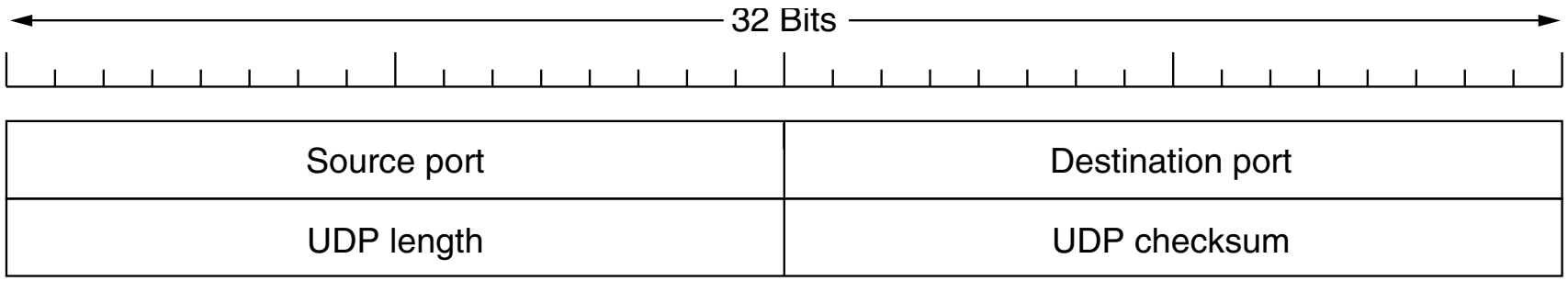
- **Objective:** To provide a connection-less transport service
  - Very little functionality beyond sending packets between applications
  - Lets applications build their own protocols on top as needed!

- Defined in RFC 768 (1981)

UDP: Segment Structure

Institute for the Wireless Internet of Things at Northeastern

- Composed by:
  - 8-byte header:
    - Source Port
    - Destination Port
    - Length: from 8 bytes to 65,515 bytes
    - Checksum: 16-byte CRC
  - Payload



UDP: Ports

- The **main advantage of UDP over just IP** is the addition of the **source and destination ports**
  - Without the port fields, the transport layer would not know what to do with each incoming packet

- The receiver of a UDP segment must have a socket bound to a specific port
  - When a UDP packet arrives, its **payload is handed to the process attached to the destination port**

- The sender of a UDP segment must have a socket bound to another specific port too, mainly to be able to receive the reply from the destination

Transmission Control Protocol (TCP)

- Formally defined in RFC 793 (1981)
  - Plus several amendments, e.g., RFC 1122, RFC 1323, RFC 2018, …

- **Objective:** To provide a reliable end-to-end byte stream over an unreliable internetwork

TCP: Service Model

- All TCP connections are full duplex and point-to-point:
  - Remember:
    - Full duplex: traffic can go in both directions at the same time
    - Point-to-point: each connection has exactly two end points
- TCP does not support multicasting or broadcasting
- A TCP connection is a byte stream, not a message stream

TCP: Service Model

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

- However, sometimes the application really wants the data to be sent immediately:
  - For example, suppose a user of an interactive game wants to send a stream of updates and it is essential that the updates be sent immediately

- To force data out, TCP has the notion of a **PUSH flag** that is carried on packets, whose original intent was to let applications tell TCP implementations not to delay the transmission

- However, applications cannot literally set the PUSH flag when they send data. Instead, different operating systems have evolved different options to expedite transmission (e.g., TCP NODELAY in Windows and Linux)

TCP: Protocol Overview

- A **TCP segment** consists of a fixed 20-byte header (plus an optional part) followed by zero or more data bytes

- The TCP software decides how big segments should be in order **to avoid packet fragmentation**:
  - It can either accumulate data into one segment or can split data from one write over multiple segments
  - Two limits need to be satisfied to avoid fragmentation:
    - On the one hand, each segment, including the TCP header, must fit in the 65,515-byte IP payload
    - On the other hand, each segment must fit in the MTU for all the links between the sender and the receiver
      - In practice, the MTU is generally 1500 bytes (the Ethernet payload size) and thus defines the upper bound on segment size
      - Path MTU discovery is used (Chapter 5 – Part 2, Slide 20)

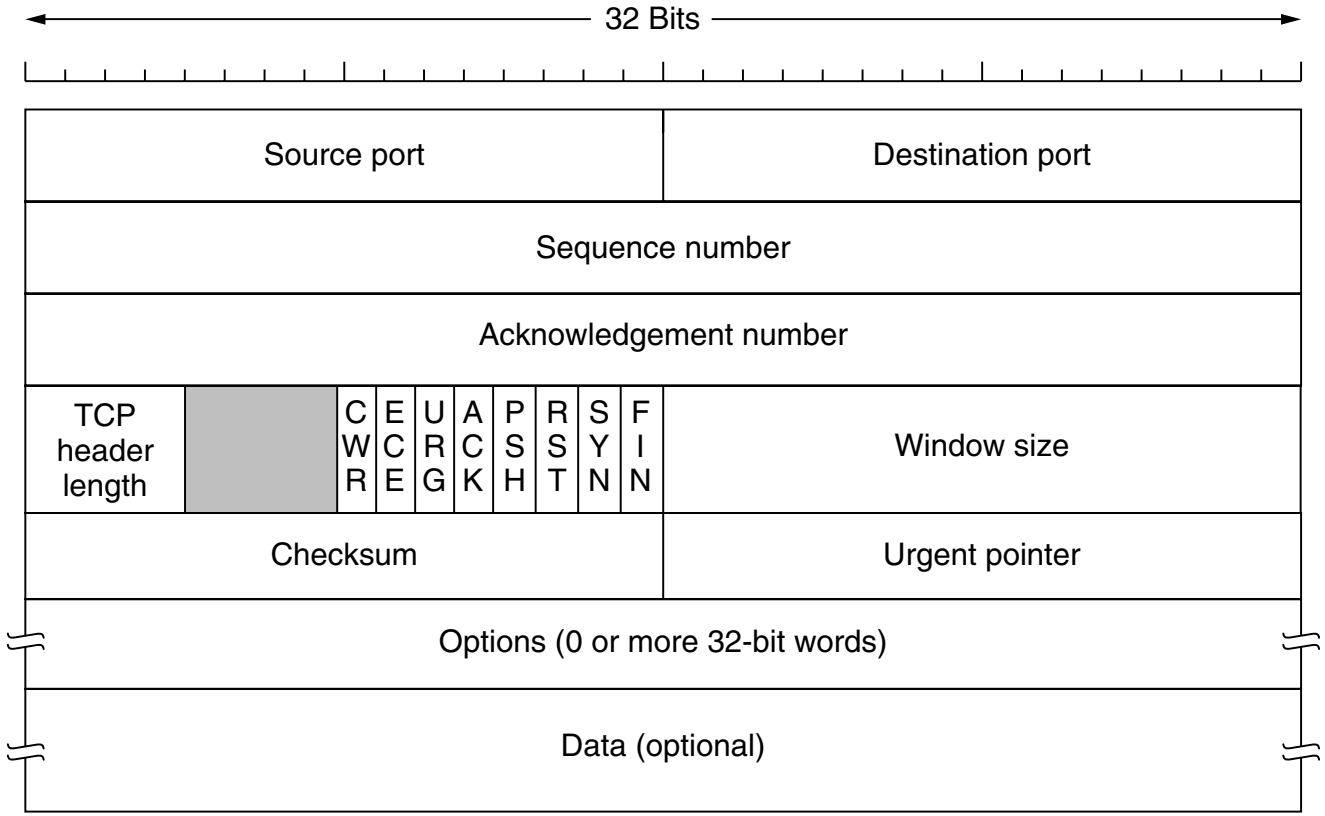
TCP: Protocol Overview

- The basic protocol used by TCP entities is the sliding window protocol with a dynamic window size:
  - When a sender transmits a segment, it also starts a timer
  - When the segment arrives at the destination, the receiving TCP entity sends back a segment (with data if any exist) and an acknowledgement number equal to the next sequence number it expects to receive and the remaining window size
  - If the sender's timer goes off before the acknowledgement is received, the sender transmits the segment again

- Some fine tuning is needed because
  - Segments can arrive out of order
  - Segments can also be delayed so long in transit that the sender times out and retransmits them
    - The retransmissions may include different byte ranges than the original transmission, requiring careful administration to keep track of the bytes!

TCP: Segment Header

- 20 byte long



TCP: Segment Header

- **Source Port**
- **Destination Port**
- **Sequence Number**
- **Acknowledgement Number:** Cumulative acknowledgement
- **TCP Header Length:** Indicates how many 32-bit words are contained in the TCP header
- 4 empty bits!
- 8 flags:
  - Explicit Congestion Notification (ECN) Echo (**ECE**): To notify the sender that it should slow down
  - *Congestion Window Reduced (**CWR**):* To notify the receiver that the sender has slowed down its transmission
  - *Urgent pointer (**URG**):* To indicate that the **urgent pointer** is active (this shows a byte offset from the current sequence number at which urgent data are to be found)
  - **ACK**: To indicate the acknowledgement number is valid

TCP: Segment Header

- Pushed Data (**PSH**): To indicate pushed data
- Reset (**RST**): To abruptly reset a connection due a host crash or similar conditions
- **SYN**: To establish connections
- **FIN**: To release a connection

- **Window size:** Indicates how many bytes may be sent starting at the byte acknowledged
  - Note that now acknowledgements and windows size are decoupled:
    - A node can say "I have properly received till byte XX, but I don't want any more data. Thanks."
- **Checksum**
- **Options:**
  - **Maximum Segment Size (MSS):** in case the receiver cannot handle large segments
  - **Selective Acknowledgement (SACK):** instead of the cumulative acknowledgement
  - **…**

TCP: Connection Establishment

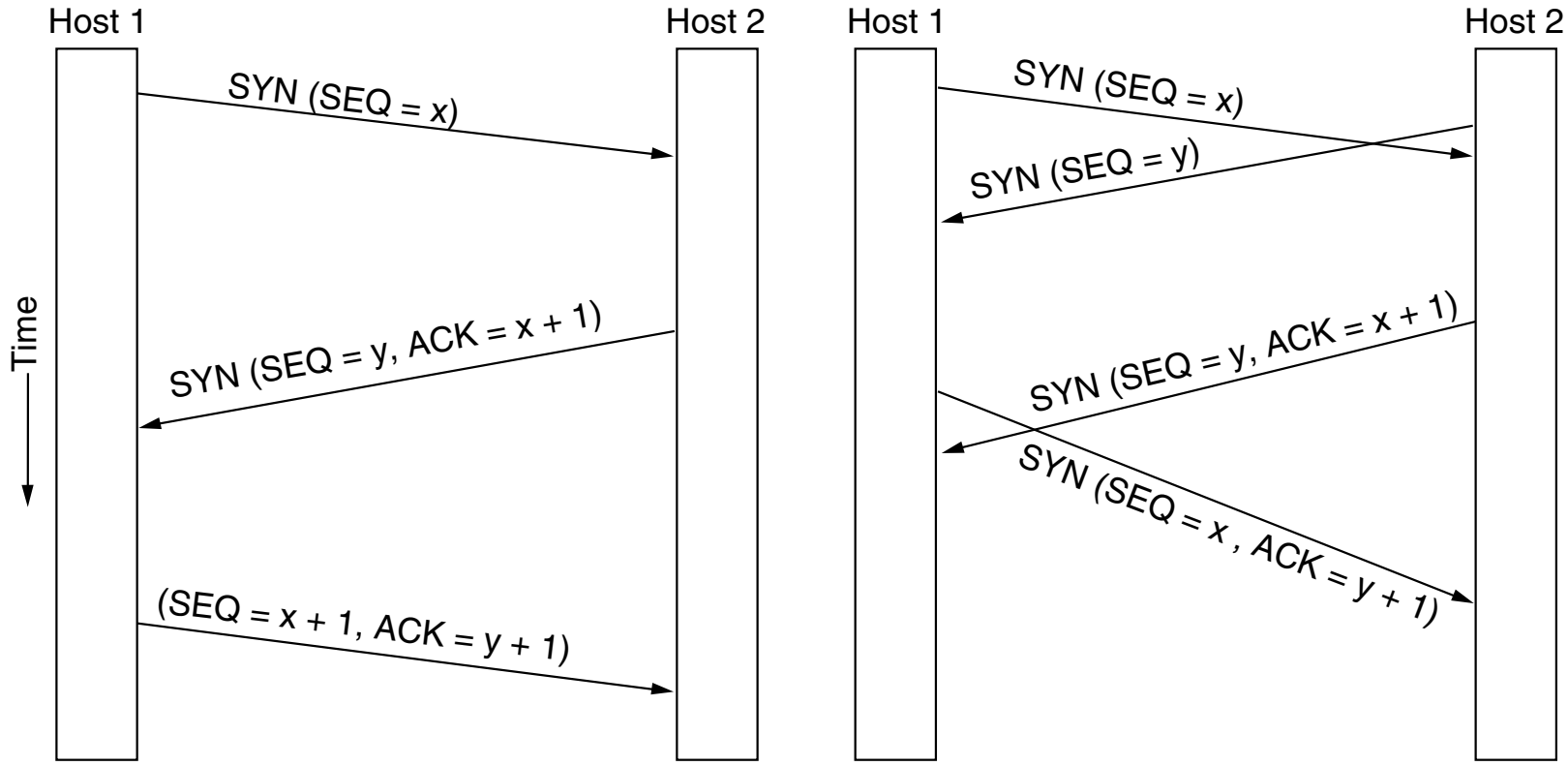
- Based on the three-way handshake explained before:
  - To establish a connection, one side, i.e., the server, passively waits for an incoming connection by executing the LISTEN and ACCEPT primitives
  - The other side, i.e., the client, executes a CONNECT primitive, specifying:
    - The IP address and port to which it wants to connect
    - The maximum TCP segment size it is willing to accept
    - Optionally some user data (e.g., a password)
  - The CONNECT primitive sends a TCP segment with the *SYN* bit on and *ACK* bit off and waits for a response

TCP: Connection Establishment

- When the CONNECT segment arrives at the destination, the TCP entity there checks to see if there is a process that has done a LISTEN on the port given in the Destination port field
  - If not, it sends a reply with the RST bit on to reject the connection
  - If some process is listening to the port, that process is given the incoming TCP segment:
    - It can either accept or reject the connection:
      - If it accepts, an acknowledgement segment is sent back

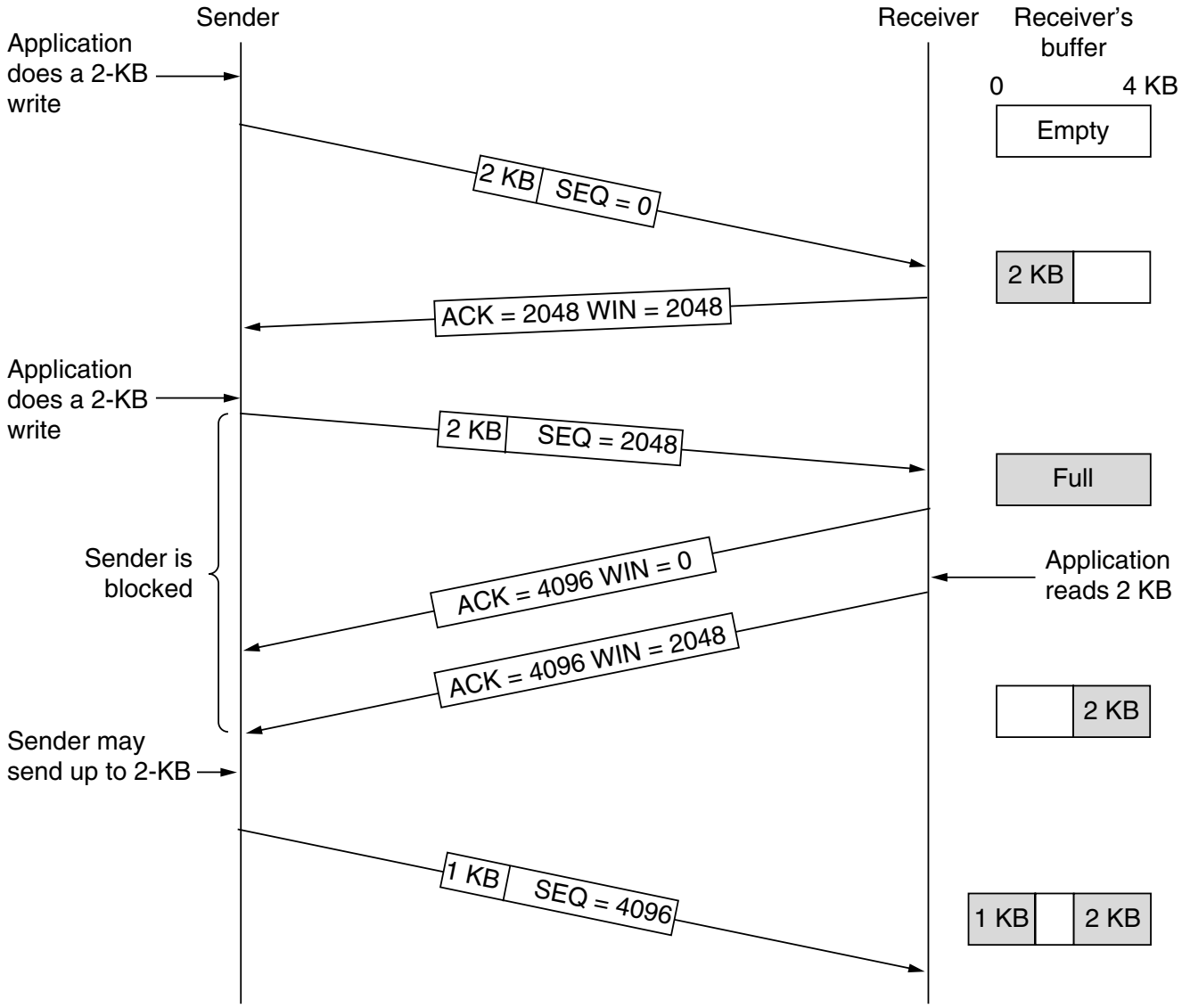
- In the event that two hosts simultaneously attempt to establish a connection between the same two sockets,
  - Just one connection is established, not two, because connections are identified by their end points
  - If the first setup results in a connection identified by (*x*, *y*) and the second one does too, only one table entry is made, namely, for (*x*, *y*).

TCP: Connection Establishment



TCP: Sliding Window

- There are two windows in TCP:
  - **Congestion window:** tries not to exceed the capacity of the network (congestion control)
  - **Receiver window:** tries not to exceed the capacity of the receiver to process data
- The details are beyond the scope of this course.



TCP: Connection Release

- Although TCP connections are full duplex, to understand how connections are released it is best to think of them as a pair of simplex connections
  - Each simplex connection is released independently of its sibling
- To release a connection,
  - Either party can send a TCP segment with the *FIN* bit set, which means that it has no more data to transmit
  - When the *FIN* is acknowledged, that direction is shut down for new data
- Data may continue to flow indefinitely in the other direction
- When both directions have been shut down, the connection is released
- Normally, four TCP segments are needed to release a connection: one *FIN* and one *ACK* for each direction
  - However, it is possible for the first *ACK* and the second *FIN* to be contained in the same segment, reducing the total count to three

Focus on IoT

Institute for the Wireless Internet of Things at Northeastern

- "Things" have unique constraints for dependable data delivery that impact the transport layer
  - Limited energy (battery powered!)
  - Limited computational resources in a node
  - Limited memory
  - Limited dependability of individual nodes
  - Transmission errors over a wireless channel

- Standard solution to increase end-to-end reliability: **Redundancy**
  - Redundancy in nodes, transmission
  - Forward and backward error recovery
  - Combinations are necessary

Constraints

- Energy
  - Send as few packets as possible
  - Send with low power → high error rates
  - Avoid retransmissions
  - Short packets → weak FEC
  - Balance energy consumption in network
- Processing power
  - Only simple FEC schemes
  - No complicated algorithms (coding)
- Memory
  - Store as little data as briefly as possible

Delivering Single Packets

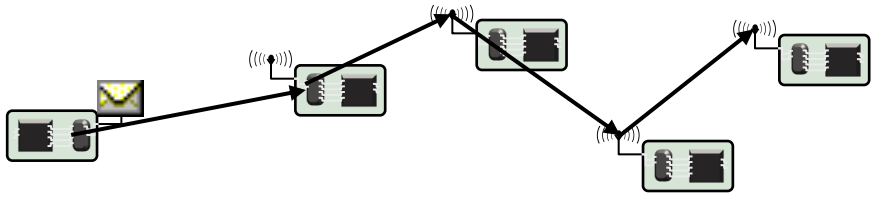
- What are the intended receivers?
  - A single receiver?
  - Multiple receivers?
    - In close vicinity? Spread out?
  - Mobile?

- Which routing structures are available?
  - Unicast routing along a single path?
  - Routing with multiple paths between source/destination pairs?
  - No routing structure at all – rely on flooding/gossiping?

Single Packet, Single Receiver, Single Path

Institute for the Wireless Internet of Things at Northeastern

- Single, multi-hop path is given by routing protocol



- Issues: Which node
  - Detects losses (using which indicators)?
  - Requests retransmissions?
  - Carries out retransmissions?

Single Packet, Single Receiver, Single Path

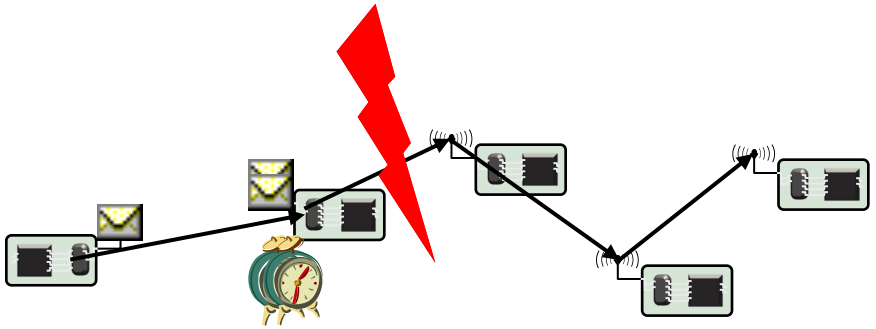
Institute for the Wireless Internet of Things at Northeastern

- Detecting loss of a single packet:
  Only positive acknowledgements (ACK) feasible
  - Negative acks (NACK) not an option
  - Receiver usually does not know a packet should have arrived, has no incentive to send a NACK

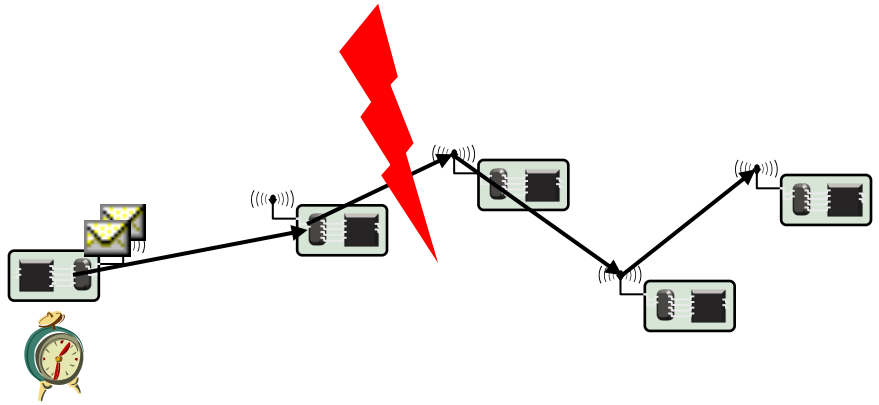
- Which node sends ACKs (avoiding retransmissions)?
  - At each intermediate node, at MAC/link level
    - Usually accompanied by link layer retransmissions
    - Usually, only a bounded number of attempts
  - At the destination node
    - Transport layer retransmissions
    - Problem: Timer selection

Retransmissions

- For link layer acknowledgements: Neighboring node



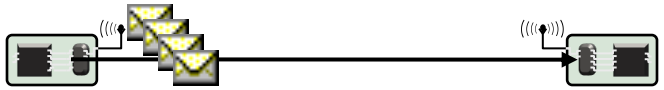
- For transport layer acknowledgements:
  - Source node → end-to-end retransmissions



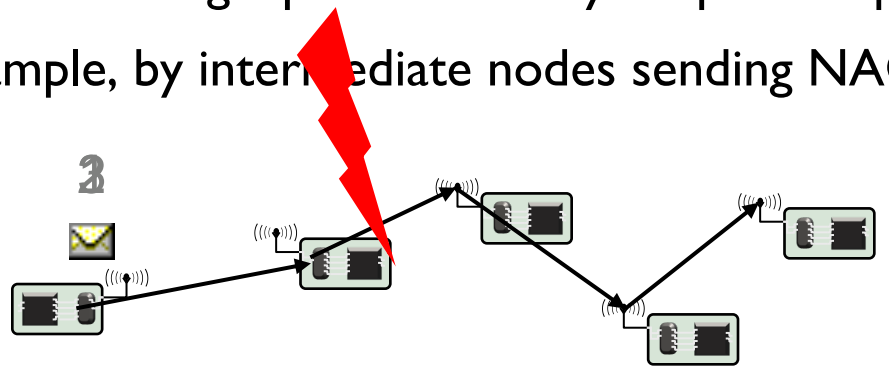
Delivering Blocks of Packets

Institute for the Wireless Internet of Things at Northeastern

- Goal: Deliver large amounts of data
  - E.g., code update, large observations
  - Split data into several packets (reduce packet error rate)
  - Transfer this block of packets



- Main difference to single packet delivery: Gaps in sequence number can be detected and exploited
  - For example, by intermediate nodes sending NACKs



Where is packet 2?

- To answer NACK locally, intermediate nodes must cache packets
- Which packets? For how long?

Many Solutions

Institute for the Wireless Internet of Things at Northeastern

- Once again, in the 90s and early 2000s, many solutions were developed for the Wireless Sensor Networks (the IoT ancestor), tackling different problems in different ways

- Examples:
  - Pump Slowly Fetch Quickly (PSFQ)
  - Reliable Multisegment Transport (RMST)
  - Event-to-Sink Reliability (ESRT)

- The reality:
  - With all the current IoT solutions, "Things" are usually one-hop away from an IP network
  - As we will see when in Module T5, almost everything goes on top of TCP or UDP

Next Steps

Institute for the Wireless Internet of Things at Northeastern

- Research paper:
  - Due on November 20

- Coming soon:
  - Homework assignment (numerical)
  - Online (open-book) quiz (multiple choice questions)