Internet of Things
IO 4041
Transport Layer

UDP for smart objects

Benefits of UDP in smart object networks

- very low overhead for both header size and protocol logic.

- Therefore both the packet transmissions and receptions consume less energy, and

- each packet has more room for application layer data.

- Since the protocol is simple, implementations typically have a very small code footprint.

UDP for smart objects

The simplicity of UDP also fits well with many smart object applications.

- Like in home automation system, temperature sensors may periodically report data.
- The sensors can use UDP to send the data and to achieve low overhead.
- Since data are sent periodically,
  - So it does not matter that individual packets may be lost
  - a new temperature reading will be sent soon enough anyway.

UDP for smart objects

UDP is well suited to traffic with low reliability demands (smart object systems).

- reliability can be provided at the application layer,
- but this increases the complexity of the application.

UDP is also well suited to applications requiring own routing mechanisms

- For these applications, routing can be implemented as an application overlay mechanism

UDP for smart objects

Drawbacks of UDP for smart object networks

- **often lose packets in transit.**
  - No recovery mechanism for lost packets.
  - Up to the application to recover from packet loss, which increases the complexity of applications that require reliability.
- **often have small packet sizes**
  - UDP does not provide any mechanism for applications to split their data into appropriately sized chunks for transmission.
  - So application must figure out appropriate packet size and adjust its packets accordingly.

Unlike UDP,
TCP provides both reliability and a mechanism to automatically limit the packet sizes sent by applications.

TCP for smart objects

Compelling properties of TCP for smart object networks

- Many smart object networks operate over links where packets can be lost,
- many applications may want to use a reliable mechanism that automatically retransmits lost packets
- Though TCP has performance problems for high-throughput data when packets are lost over wireless links,
- BUT for many smart object networks
  - high throughput is not the primary objective and
  - reliable delivery of data is more important
- ❖ Moreover, such networks often interoperate with existing systems where TCP is very widely adopted,
  - So the ability to directly communicate with existing systems speaks in favor of TCP.

TCP for smart objects

The small packet sizes in many smart object networks require

- the packets to be kept small enough to fit,
- but large enough to effectively use available resources.
- The **TCP MSS** option is very useful both
  - for memory-constrained systems and
  - for systems constrained by a small packet size, such as systems running over wireless 802.15.4 networks.
- The TCP MSS option provides a way to set a small packet size for all TCP packets sent over the network

Whereas UDP, no mechanism for limiting the size of sent and received packets exists.

TCP for smart objects

As high-throughput performance is not strict requirement in many smart object networks

several mechanisms in TCP are not needed such as the sliding window algorithm and delayed ACKs.

TCP for smart objects

Two limiting factors in severe resource constraint for TCP implementation

I. memory-constrained TCPs do not implement the sliding window mechanism
   - ✓ i.e., a TCP sender cannot have more than one TCP packet for each active TCP connection in the network at any given time

II. TCP delayed ACKs (widely applied by TCP) reduce the throughput
   - ✓ intended to reduce the amount of ACK packets sent over a TCP connection.
   - ✓ incoming TCP data are not ACKed immediately
   - ✓ the host waits for a short time, usually 200 ms, before sending the ACK

TCP for smart objects

During wait time (in delayed ACK), another TCP segment may arrive

- ✓ If it arrives, the ACK is sent immediately
- ✓ If not arrives, the ACK is sent after 200 ms
- ✓ This effectively reduces the amount of ACKs by half for a busy TCP connection
- ✓ **For a constrained TCP sender, who only sends one TCP segment at a time**,
  - ✓ the delayed ACK mechanism may significantly reduce the throughput
  - ✓ turning off the delayed ACK mechanism at the receiver to avoid this problem

TCP for smart objects

During wait time (in delayed ACK), another TCP segment may arrive

- ✓ If it arrives, the ACK is sent immediately
- ✓ If not arrives, the ACK is sent after 200 ms
- ✓ This effectively reduces the amount of ACKs by half for a busy TCP connection
- ✓ **For a constrained TCP sender, who only sends one TCP segment at a time**,
  - ✓ the delayed ACK mechanism may significantly reduce the throughput
  - ✓ turning off the delayed ACK mechanism at the receiver to avoid this problem

Transport Layer Protocols for WSNs

Reliable Multi-Segment Transport (RMST) Protocol

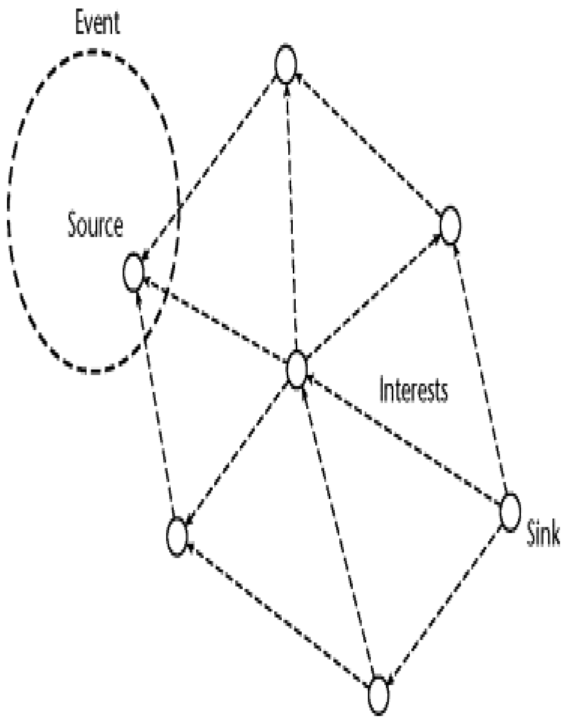
- Provides end to end reliability
- designed as a filter that could be attached to the **directed diffusion protocol**

Directed diffusion is **a query-based protocol**

▪a query is flooded in the network by the sink where multiple routes are established between the sink and source.

▪The sink reinforces one of the paths and receives data in a shorter interval through this reinforced path.

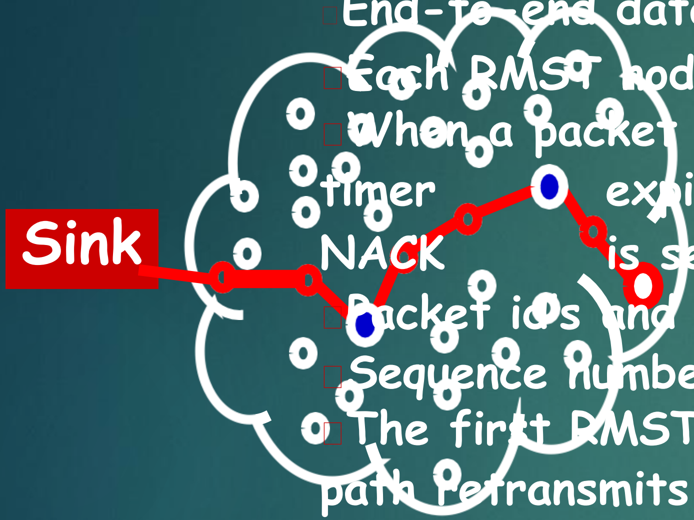
▪So packets of a flow follow the same path unless there is a node failure



Reliable Multi-Segment Transport (RMST)

F. Stann and J. Heidemann, "RMST: Reliable Data Transport in Sensor Networks," Proc. IEEE SNPA'03, May 2003, Anchorage, Alaska, USA

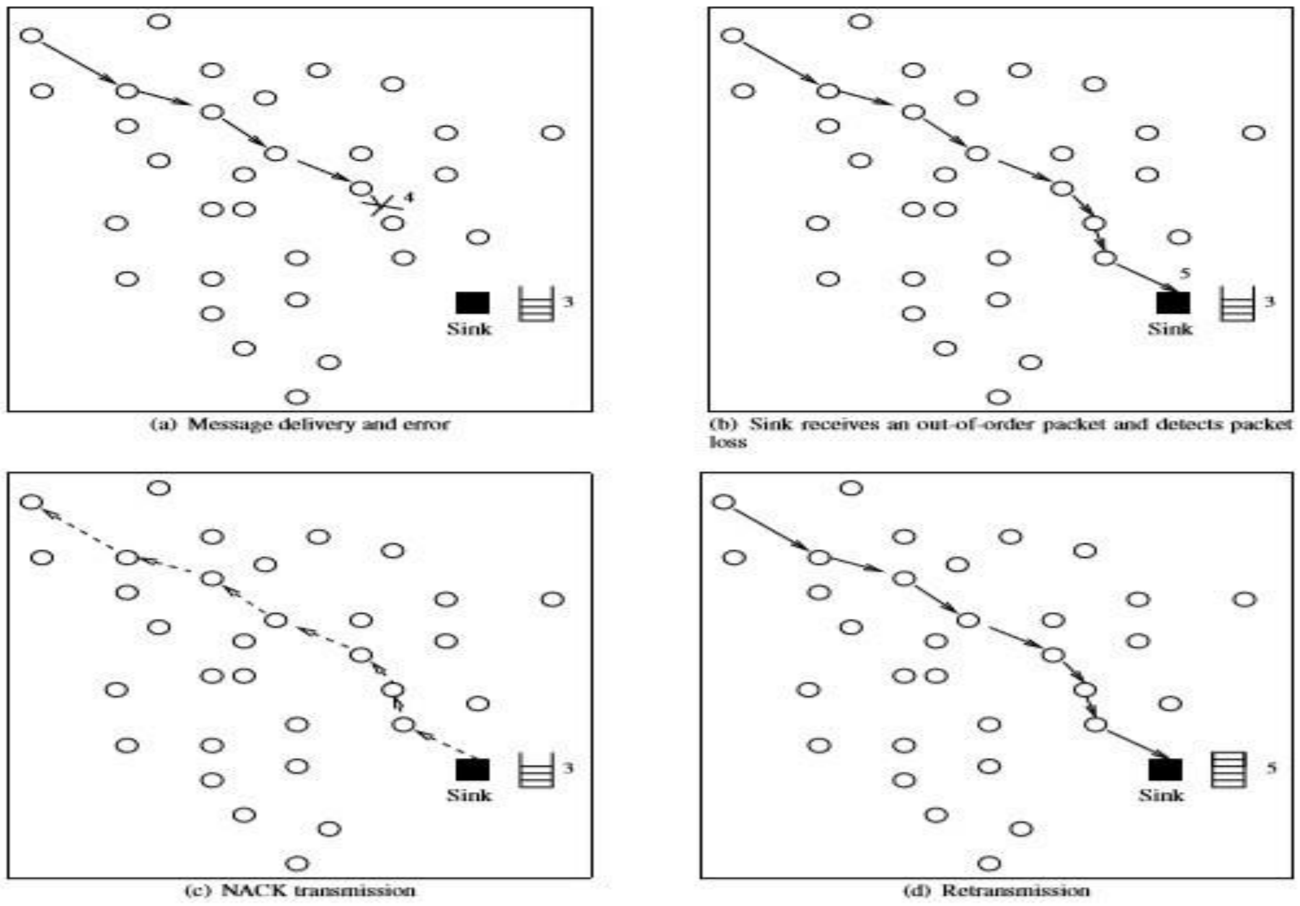
- End-to-end data-packet transfer reliability
  - Each RMST node caches the packets
  - When a packet is not received before the so-called WATCHDOG timer expires, a NACK is sent backward
    - Packet id's and number of packet
    - Sequence number of holes
  - The first RMST node that has the required packet along the path retransmits the packet



- ● RMST Node
- ● Source Node

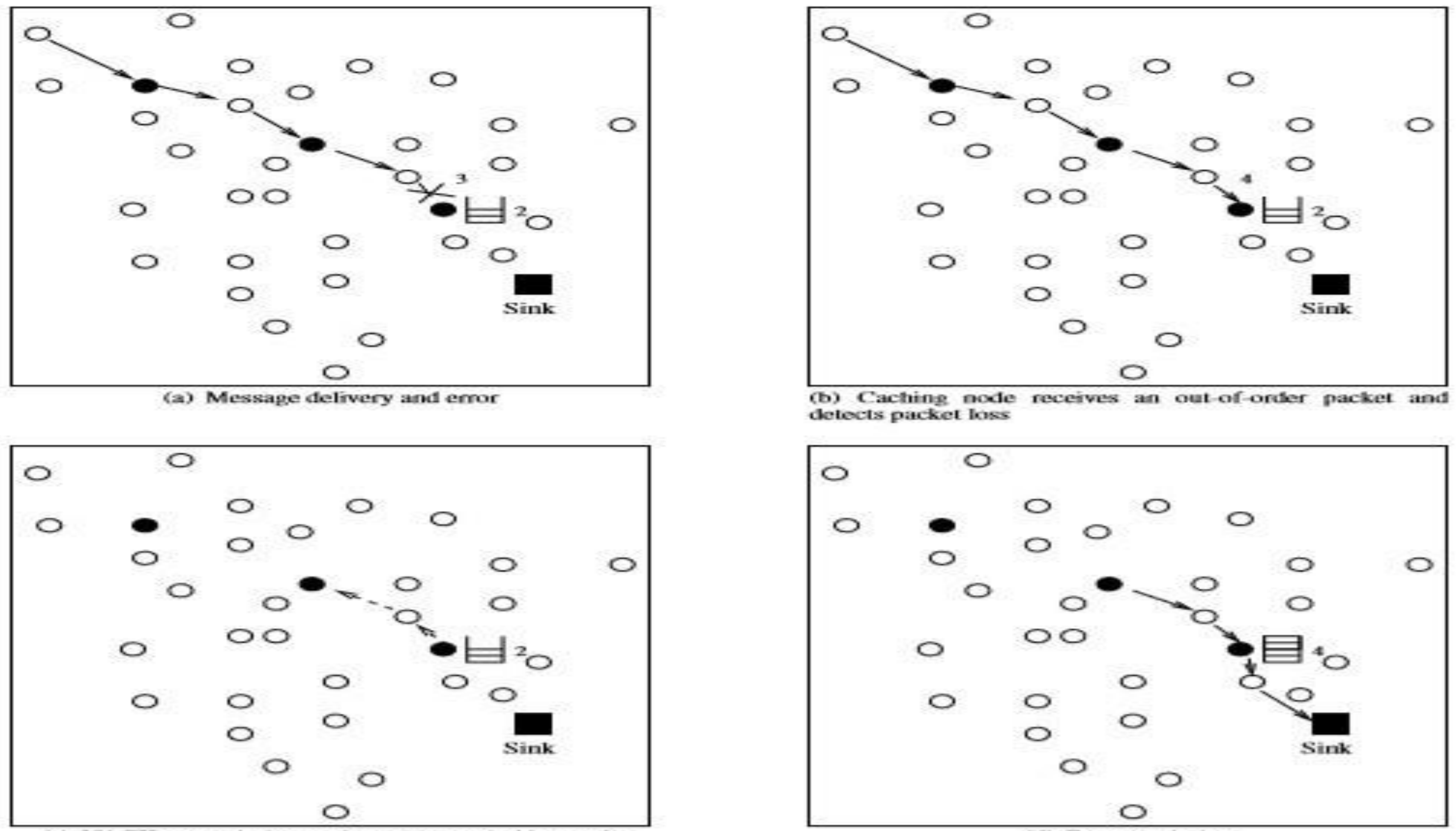
- In-network caching brings significant overhead in terms of power and processing
- Relies on Directed Diffusion Scheme

Non Caching Example:


4
Sink
3
(a) Message delivery and error
5
Sink
3
(b) Sink receives an out-of-order packet and detects packet loss
Sink
3
(c) NACK transmission
Sink
5
(d) Retransmission

Error recovery with RMST in non-caching mode.

Caching Example:



(a) Message delivery and error

(b) Caching node receives an out-of-order packet and detects packet loss

(c) NACK transmission to the upstream caching node

(d) Retransmission

Error recovery with RMST in caching mode. Black nodes represent the caching nodes.

Non Caching Example:

- Error recovery in non-caching mode where a sensor node is trying to transmit a series of packets to the sink through the multi-hop route.
- The sequence number of the last received packet at the sink is shown in each case.
- In non-caching mode, end-to-end retransmissions are performed to provide reliability.
- Figure (a), packet 4 is lost before reaching the sink.
- The sink can only recognize the packet loss after it receives packet 5 (Figure (b)).
- Then, the sink transmits a NACK packet back to the source node as shown in Figure (c).
- The lost packet is then retransmitted to the sink through the multi-hop route as shown in Figure (d)

Caching Example:

- In caching mod, certain sensor nodes are assigned as caching nodes (denoted as black nodes in Figure) on the reinforced path from the sensor node to the sink.
- In addition to the sink, loss packet detection is also performed at the caching nodes.
- Similar to the non-caching case, the loss of packet 3 in Figure (a) can only be detected by the caching node after receiving packet 4 as shown in Figure (b).
- Then, the caching node transmits a NACK packet back to the source node as shown in Figure (c).
- As shown in Figure (d), the first caching node with the missing packet on the reverse path replies and the packets are transmitted to the sink in order. I
- f the packet cannot be found in one of the caching nodes, the NACK packet is propagated until it reaches the source node.