Aalto-yliopisto

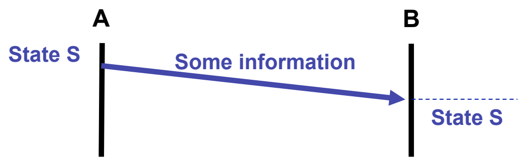
Reliability

Protocol Design – S-38.3159

Aalto-yliopisto

Basic Purpose of a Protocol

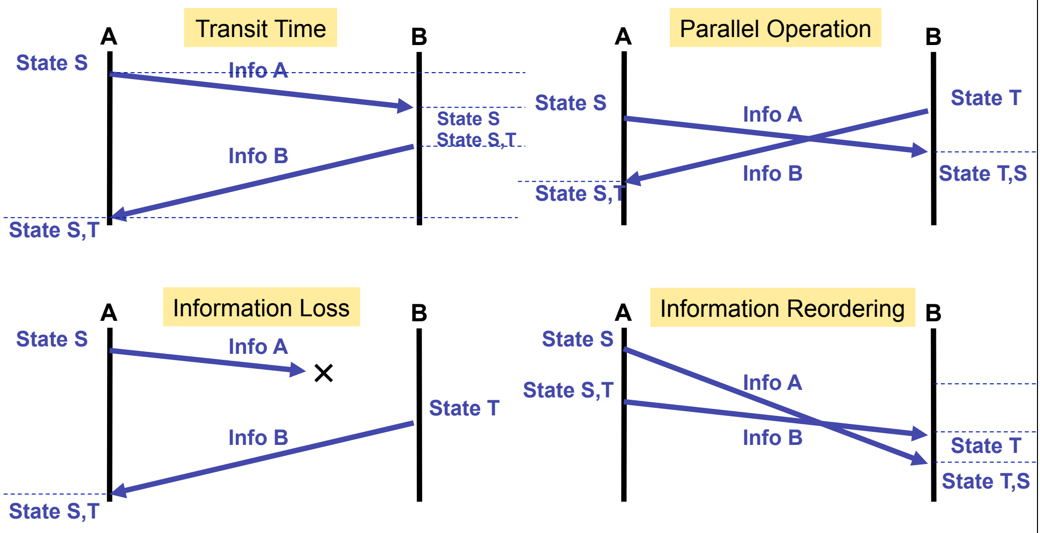
- Synchronize state information across two or more nodes
- State can be anything
  - Some data item
  - Existence and parameters of a communication relationship
  - Parameters for and result of an operation
  - Contents of a database or file
- State synchronization should be “reliable”…
  - To be achieved with a minimal number of message exchanges



Distributed Systems Fundamentals

- In a distributed system, each node has their own view of reality
  - Information takes time in transit
  - Not all information arrives intact
  - Information does not arrive in order
- There is no global view
- There is no global concept of “simultaneous”
- Entities are independent and may operate in parallel
  - Uncertainty what the other peer(s) do or believe at a given point in time
- Synchronizing entities require effort (=overhead)
  - The closer the synchronization, the higher the overhead

Distributed System Fundamentals (2)

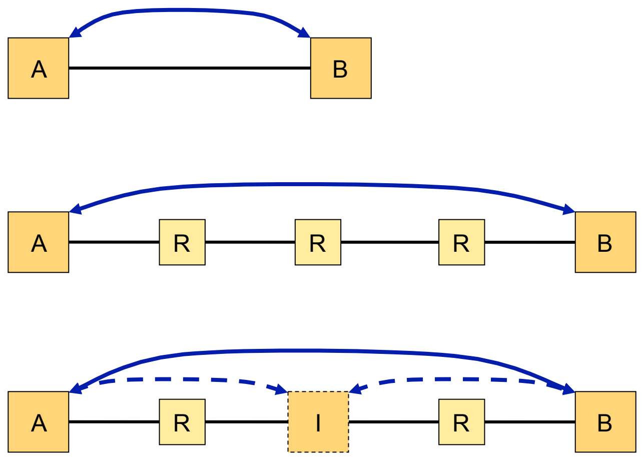


Some System Setup Alternatives

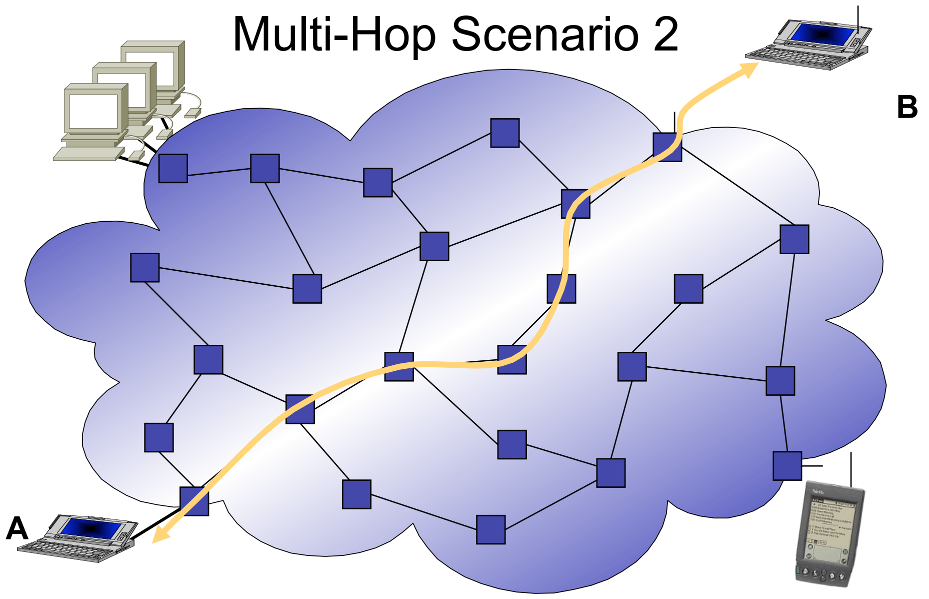
1) Direct link

2) Multiple hops

3) Multiple hops with intermediary support

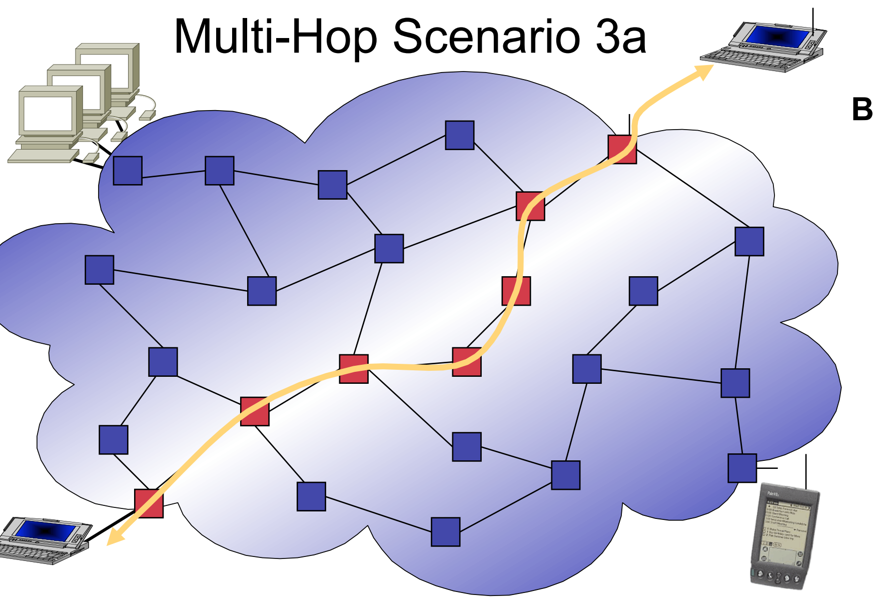


Multi-Hop Scenario 2



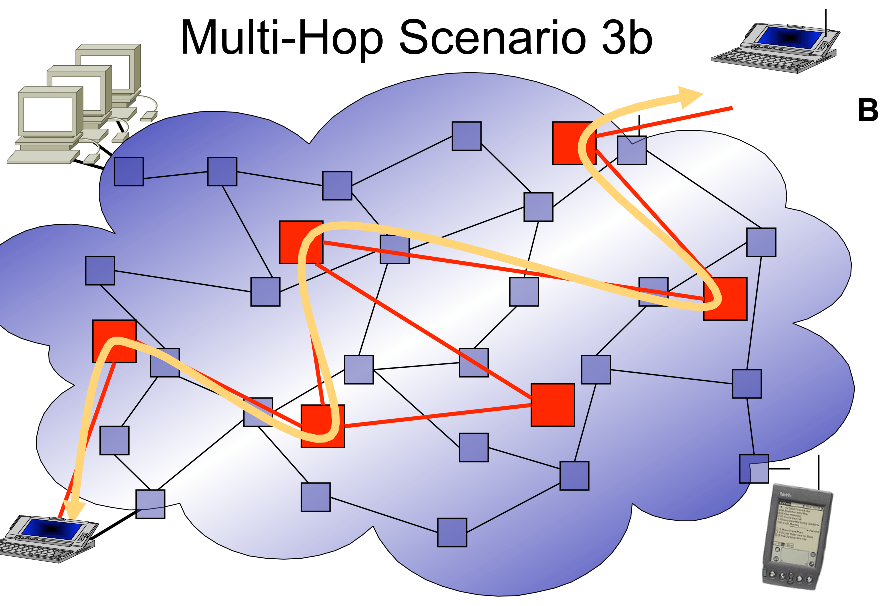
Aalto-yliopisto

Multi-Hop Scenario 3a



Aalto-yliopisto

Multi-Hop Scenario 3b



What can go “wrong”?

- Effects of a link
  - Bit errors (individual vs. bursty bit errors)
  - Frame losses (individual vs. bursty losses) ➔ packet losses
  - Latency (medium access, physical propagation, and serialization delay)
  - Frame reordering (e.g., due to individual losses and retransmissions or multiplexing)
- Effects caused in a router or due to routing
  - Packet losses (even distribution, burstiness – depends on queuing scheme)
  - Packet corruption
  - Packet duplication (typically due to routing along different paths)
  - Packet delay (varies depending on queue size, i.e., offered load)
  - Packet reordering (typically due to load sharing along different paths)
- Errors and other effects in the network
  - Routing loops or black holing (causing packet loss)
  - Router crashes or link unavailability (causing temporary unreacheability and variation in QoS, packet loss)
  - Route changes (due to failures, for load balancing, etc.) causing variation is path characteristics
  - Memory or software errors in routers
  - Unidirectional or otherwise asymmetric links
  - Congestion (from legitimate traffic or DoS: causing packet loss and latency)

What can go “wrong”? (2)

- Effects in the end system
  - Packet losses due to buffer overflow (too many interrupts, CPU overload, …)
  - Application failure or crashes
  - Malfunctions (partial or complete, malicious or accidental)
  - Failures (silent or reported/observable, byzantine, …)
  - Overload (DoS or just plain heavy load)
- Effects due to mobility
  - Rerouting leads to different latencies (and other transmission characteristics)
  - Rerouting may lead to packet loss, packet bursts, reordering
  - Temporary unavailability
  - (possibly changes in identification)
- And other things you may and those you may not expect…

Aalto-yliopisto

Reliability is Probabilistic

- Variety of mechanisms available to deal with things that go wrong to improve reliability
  - Checksums, CRCs, MACs to detect bit errors or frame errors in packets
    - Avoid processing an incorrect frame (which may lead to confusion in the state machine)
  - Sequence numbers to detect missing packets
- Implicit assumption: errors are of temporary nature
  - E.g., retransmissions will work after several attempts
  - Depending on the error probability this may be sooner or later
  - Protocols define their own “patience” (aka timeout), i.e., how long or how often they are willing to try
- Most reliable protocols fail if the error condition persists long enough
- A reliable protocol need not fail if it just tries long enough
  - Even if peer breaks and the communication context is lost
    (in which case this would need to re-established, which will take even longer)

Aalto-yliopisto

Reliability is a Tradeoff

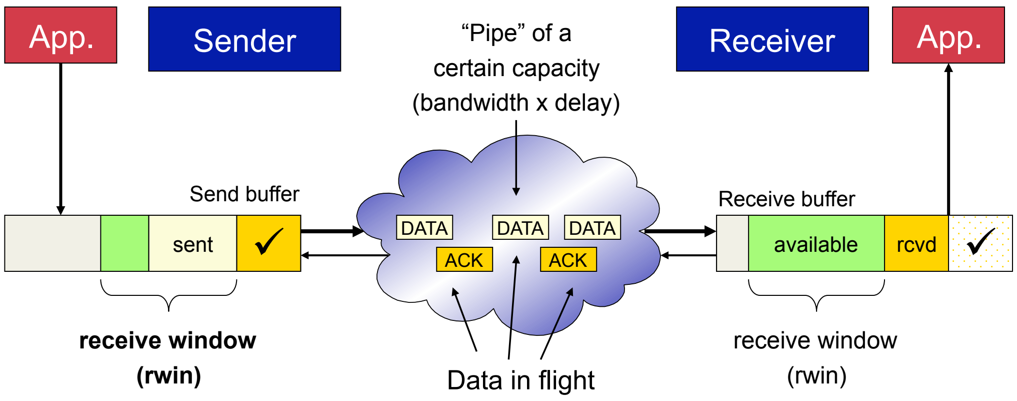
- Reliability (probability) vs. delay
- Reliability (probability) vs. overhead
  - Processing, bandwidth consumption, local state, …
  - Efficiency depends on reliability mechanisms in use
  - Probability depends on reliability mechanisms in use
- Reliability mechanisms chosen depending on
  - Application and its semantics
  - Operational environment (types of errors, error/loss rate, RTT, b/w, …)
  - Communication setup (including number of peers)
  - Tolerance with respect to delay, lost or corrupted contents, etc.

Reliability Mechanisms

Some Questions for Reliability Protocols

- What is the overhead incurred?
- What type of overhead is incurred?
  - More bits per packet? More packets? …?
- When is the overhead incurred?
  - Always vs. only in case of failures?
- What type of errors to deal with?
- How much does the sender (want to) know about the receiver(s)?
  - Reception status: (when) did data really arrive (and can a buffer be freed)?
- How many receivers can the protocol support?
  - How heterogeneous can the receiver group be?
- What does the achievable performance depend upon?

Sample Communication Model: TCP

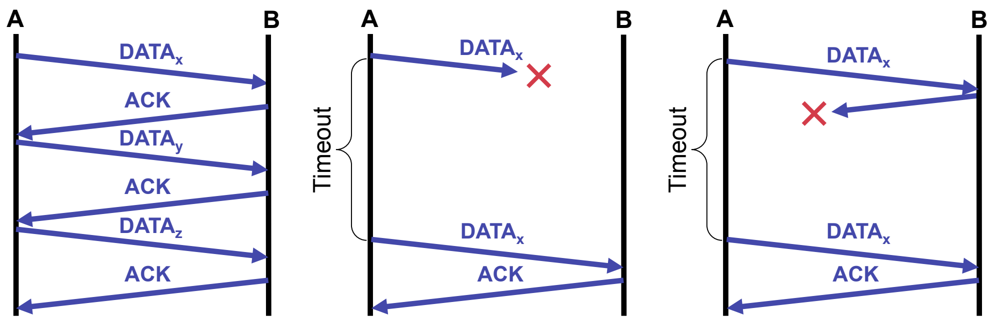


Dealing with Ordering and Overload

- Ordering: Sequence numbers (or timestamps)
  - Sequence numbers (count messages, packets, bytes)
  - Issue: avoid wrap around in fast networks
- Overload in the endpoint
  - Flow control
    - Typical windowing protocols (using seq numbers): receiver reports available buffer space
    - Issue: update frequency and ability to "keep the pipe full"
  - Rate control
    - (Predetermined) agreement between receiver and sender
    - May be updated (occasionally)
- Overload in the network: drop packets
  - Congestion control ➔ later
  - Rate control peered with resource reservations
    - Allows to influence the drop probability and delay in favor of the application
    - Reliability mechanisms need to be applied nevertheless

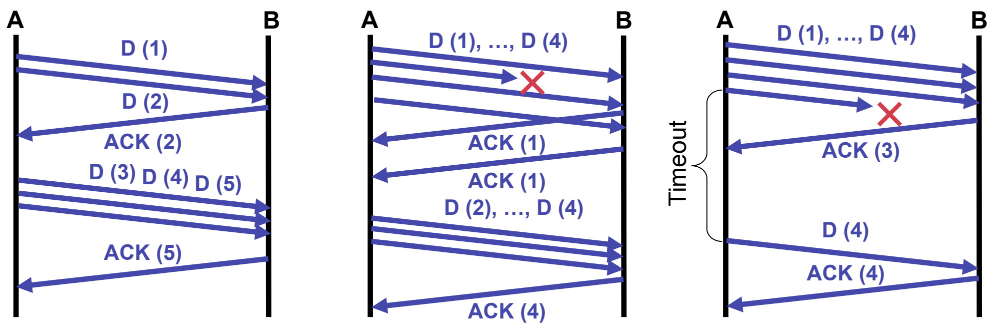
1. Simple Lock-Step Protocol

- Send data and wait for acknowledgement
- Timeout to trigger retransmission
- Trivial but very limited
- Example: Trivial File Transfer Protocol (TFTP)



2. Cumulative ACK with Go-back-N

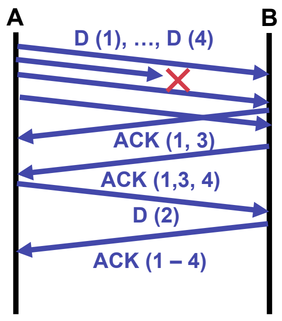
- Window-based mechanism allows multiple outstanding packets
  - constrained by sequence number space and buffer size
- Timeouts or out-of-order reception trigger retransmissions
- Variants: HDLC (LAPB/D/F), X.25 layer 3, plain old TCP, …



Aalto-yliopisto

3. Selective Acknowledgements

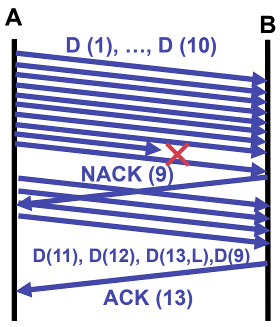
- Window-based but explicit acknowledgment of received packets
- Receiver keeps out-of-order packets (e.g., TCP SACK)

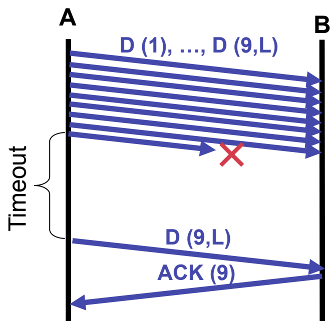


Aalto-yliopisto

4. Simple NACK Protocol

- Optimistic assumption: packets will arrive
  - Report only failures: negative acknowledgement
- Specific mechanisms needed for last packet (e.g. ACK)
- Specific mechanisms needed for flow control and buffer mgmt

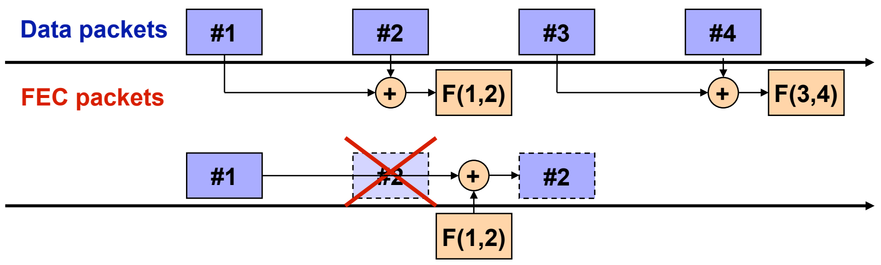




5. Forward Error Correction (1)

- Basic assumption: errors will occur
  - Increase reception probability up front:
  - Send packets + *redundancy* packets
- Simple XOR-based (parity) FEC
  - P_fec = P1 XOR P2 XOR P3 XOR … XOR Pn
- More complex FEC: e.g., Reed-Solomon codes, fountain codes, …
  - Generate N packets out of K packets: copes with losing up to N–K packets
- Trading off overhead for delay and feedback
  - No need to wait for a NACK or a timeout

Issue: Increases bandwidth requirements

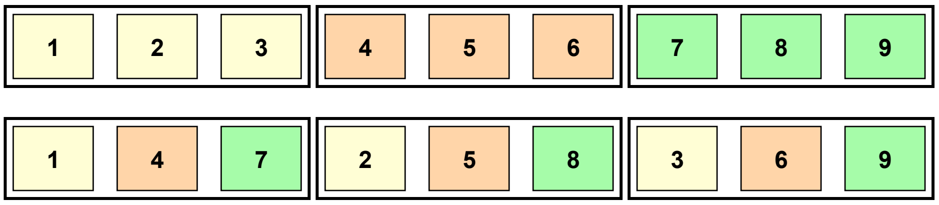


5. Forward Error Correction and TCP

Aalto-yliopisto

6. Forward Error Correction (2)

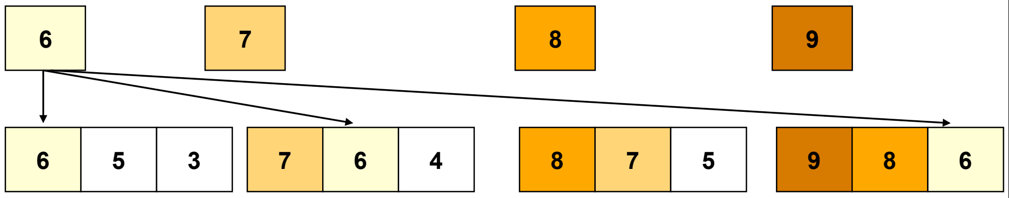
- Interleaving
  - Make simple FEC schemes work better with burst losses
- Distribute packets or packet contents for transmission
  - Avoid consecutive packet erasures in case of (burst) losses
  - Avoid loss of large consecutive data portions in case of single packet losses
- Drawbacks
  - Re-ordering causes additional delay at the receiver
  - Increases buffer space requirements



Aalto-yliopisto

7. Forward Error Correction (3)

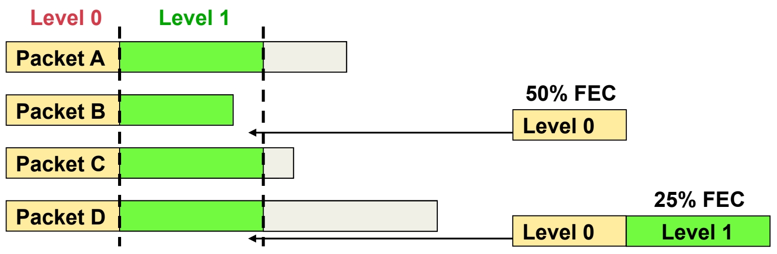
- Application-specific FEC
- Example: Fully redundant transmission
  - Primarily suitable for small pieces of information
- Repeat complete pieces of information in other packets
  - Adjacent or spread out
  - Maintains the packet rate but increases data rate
  - Dependent on regular packet transmission



Aalto-yliopisto

8. Unequal Error Protection

- Observation: not all parts of a packet are equally important
  - Beginning of packet contains headers/parameters, more relevant contents
  - Holds for both audio and video
- Uneven Level Protection (ULP)
  - Create independent parity packets for different parts of packets
  - Allows for selectively more overhead for the more important parts



- Related thoughts: partial checksums
  - Live with bit errors in the less important parts (rather than dropping a packet)

Aalto-yliopisto

9. Network Coding

- FEC: Source coding
  - Redundancy (e.g., parity packets) created at the sender
  - But: may place unnecessary on large parts of a path
  - And: considers only a single communication relationship (flows)
- Network coding: different approach to reliability
  - Collecting evidence about packets
  - Not necessarily the plain packets themselves
  - May merge packets from different flows
  - Linear combination of any number of input packets
  - Simplest case: XOR
  - Receivers collect linear combinations and decode packets as soon as they can
  - May improve performance, but may require some degree of redundancy

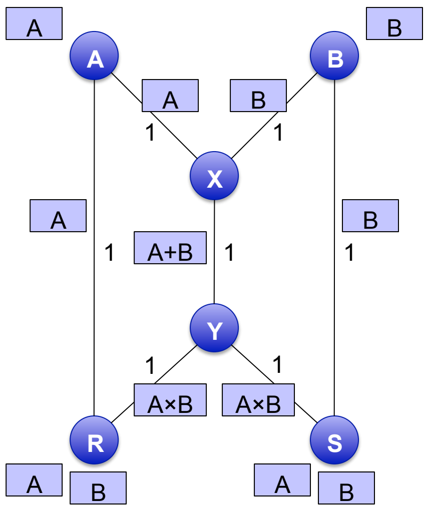
Aalto-yliopisto

9. Network Coding Example

- Network coding may reduce the delivery time and link utilization
- But routers and end systems need to conspire

Areas of application

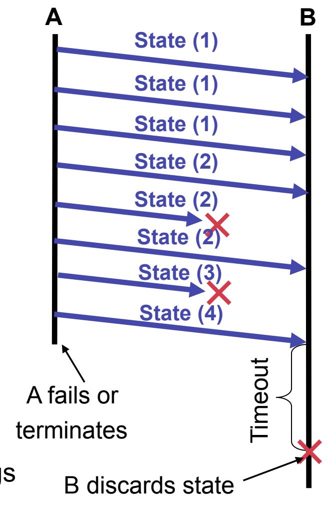
- Flooding/replication-based routing
  - E.g., in sensor networks, DTNs
- Link layer for WLANs
  - Optimizing utilization of the wireless link
  - SIGCOMM 2007 (“XORs in the air”)
- Physical layer
  - SIGCOMM 2008



Aalto-yliopisto

10. Soft State

- Reliability is typically about “hard state”
  - Explicitly created and successful creation is confirmed
  - Needs to be explicitly changed or removed
- Alternative: “soft state”
  - State is created upon packet reception
  - Needs to be refreshed periodically
  - Times out otherwise
    - Disappears automatically in case of peer failure
  - Feedback may be provided
    - E.g. Negative if state creation or modification fails
  - Issue: request or response lost vs. operation successful
  - The sender never really knows!
- Workable for small piece of information
  - May or may not change
- Examples: RSVP, some routing protocols, watchdogs



Issues with Reliability

- Shared state needed between sender and receiver
  - Receiver window, sequence number, last acknowledgement, timeout, …
  - Implicitly provided at connection setup time for connection-oriented communications
  - What about stand-alone transactions?
    - Messages need to be self-contained
    - All responsibility is with the sender
      (since the receiver does not even know that communication is imminent)
- Initialization is a potential for Denial-of-Service (DoS) attacks
- Timeout: choosing proper values
- Overhead: choosing the right combination of mechanisms
- Ideal: adapt everything dynamically to the (changing) environment

Reliable Transport Summary (1)

- State creation (aka Connection Setup)
  - N-way handshake (TCP: 3-way, SCTP: 4-way, other: 2-way)
  - Create shared state at senders and receivers
  - Issue: Denial-of-service attacks
- Error detection
  - CRC for bit errors
  - Sequence numbers against packet losses
  - Alternative naming schemes for data (name each piece of data unique)
- Error correction
  - Positive or negative acknowledgements, FEC, soft state, application-specific
  - Timeout + retransmissions
  - Different mechanisms can be combined

Aalto-yliopisto

Reliable Transport Summary (2)

- Ordering
  - Sequence numbers (or other references), buffering at the receiver
  - Optional in some cases (e.g. SCTP, TCP urgent data)
- Flow control
  - Sliding window mechanism (explicit setting of window size)
  - Implicit flow control (delayed ACKs): not relevant in the Internet
  - Rate control
- Reliability =
  Error detection + error handling (+ ordering) + flow control
  - There is no such thing like reliable communications
    - Bit errors, packet losses and network partitioning may not be repairable
    - Peers are notified of communication failures (e.g. connection teardown)
  - Degree of reliability defined by probability of communication failure

Aalto-yliopisto

Reliable Transport Summary (3)

- Congestion Control
  - Avoiding losses due to network overload
  - TCP-style mechanisms: quick response to congestion, high variation
  - Rate-based mechanisms (e.g. TFRC): slower adaptation, smoother
  - To be discussed later

Issues with Group Communications

- Potentially redefines the semantics of reliability
- One-to-many (single sender) vs. many-to-many (multiple senders)
  - Need not be IP multicast: transport/application layer replication (overlays) suffice
- “Connection” semantics: When has a “connection setup” succeeded?
  - When all intended members have joined?
  - When a quorum of intended members have joined?
  - When a certain subset of the intended members have joined?
- How does “connection setup” work?
  - Contact peers out of band? (how to make someone join a group…)
- Orderly “connection” release can be signaled in-band
- What are failure criteria for “connections”?
  - If any one member fails?
  - If a quorum of members is no longer available?
  - If any of or all of a certain subset of members fails?
- Can/should unicast-derived transport layer semantics be applied?
  - Reliable multicast semantics much more dependent on the application!

Error Detection

- Checksum (CRC) against bit errors
  - Similar to unicast transport
- Sequence numbers to detect packet losses
  - Multi-sender case: per sender sequence numbers
    - e.g. pairs of (transport address, sequence number)
    - Requires additional state in receivers

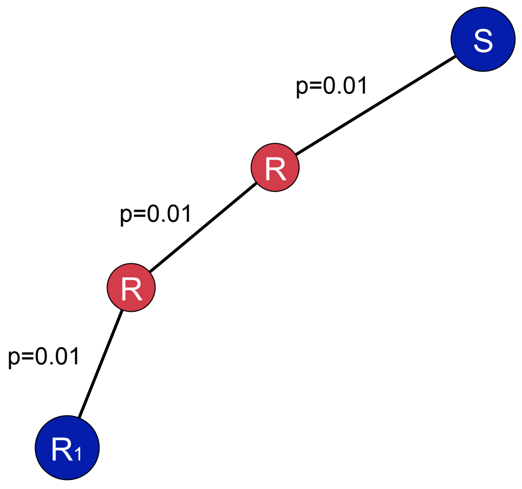
Aalto-yliopisto

Error Correction (1)

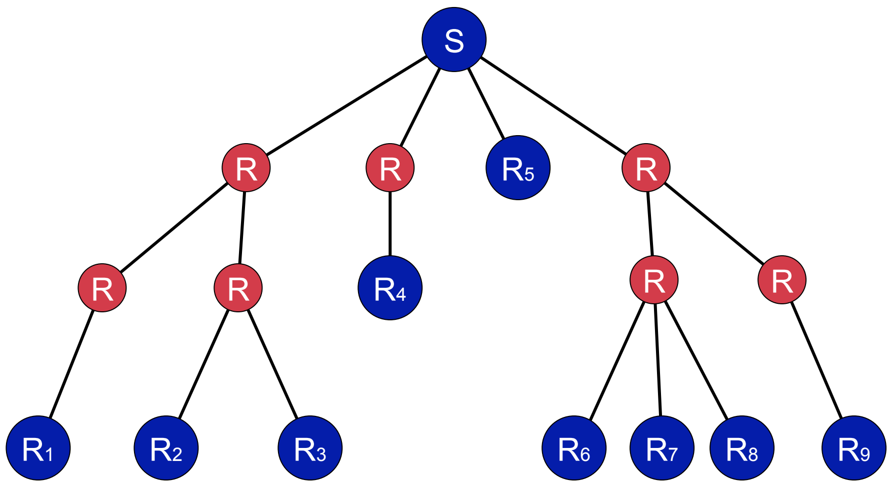
- Positive acknowledgements do not scale! (usable for small groups only)
  - ACK implosion problem at the sender
  - Different approaches needed
- Negative Acknowledgements (NACKs)
  - Cumulative or selective NACKs
  - Issue: when to release buffered data at the sender
    - Tradeoff between reliability and buffer size
  - Issue: hard to determine final state at the receivers
  - Issue: NACK implosion in case of correlated losses
- Retransmissions
  - Via multicast or via unicast
  - From the sender or some other receiver (router assist?)
- Extensive use of FEC and network coding mechanisms

Aalto-yliopisto

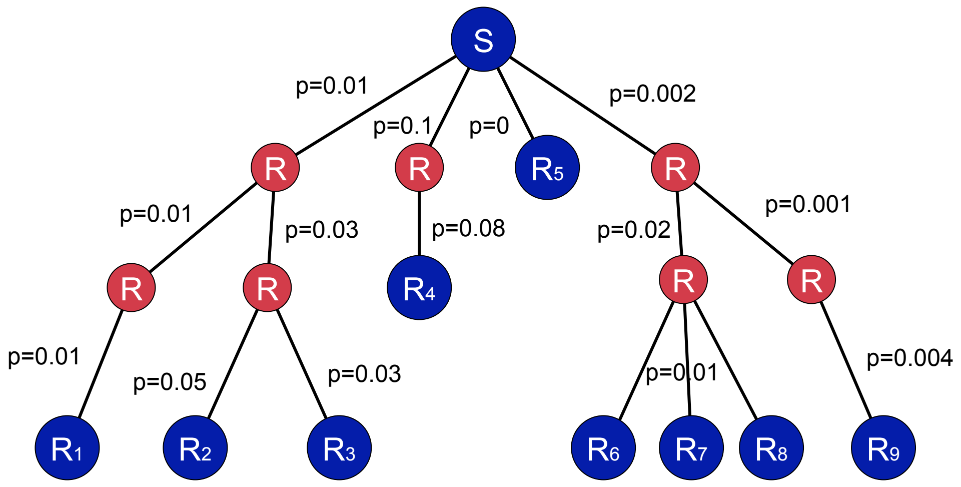
Unicast Topology: Sender and Receiver



Multicast Topology: Senders and Receivers



Multicast Topology: Senders and Receivers



Error, Flow, and Congestion Control

- A sender is supposed to throttle its transmission rate to match reception capabilities of the receiver and the network path to it.
- Which receiver?
  - All receivers?
  - A certain (subset of) receiver(s)?
  - A quorum of receivers?
- Adjusting to the worst receiver will inevitably stall the transmission
  - Compromises needed
    - Bad receivers drop out, NACKs from bad receivers are not honored, ...
  - Group communication parameters used to define minimum requirements

Reliability

- Again: reliability is probabilistic!
  - Depends on many factors
    - Packet losses, their pattern and correlation, congestion on the path
    - Buffering at the sender and time window available for retransmissions
    - FEC and other transport parameters
  - Individual vs. group reliability
- Sample reliability semantics:
  - A receiver will receive packets after joining a group and before leaving
  - The receiver will receive packets ordered per sender
  - The receiver will most likely receive all packets
  - The receiver will be notified about each packet missed
  - The receiver will be forced to leave the group if reception rate drops under a certain threshold

Ordering

- Per sender ordering trivial
  - Individual sequence numbers
- Multi-sender ordering more difficult
  - Different semantics conceivable
  - Often pushed to the application layer for efficiency
- Causal ordering
  - All dependent messages are delivered to all receivers in the same order
    - Msg B depends on Msg A if Msg A was received at a host before B is sent by this host
  - Uses message sequence vectors with one entry per node (limited scalability)
- Global ordering
  - All messages are delivered to all receivers in the same order

New Issues

- Scalability
  - What group sizes does a multicast transport protocol support?
- Atomicity
  - Did all the receivers receive the data?
  - Combination with ordering
- Partitioning and recovery
  - Network topology changes may lead to a group being split
  - Which of those parts survives?
  - What happens if partitions merge, i.e. the group is being joined together again?

Aalto-yliopisto

Relaxing Reliability Requirements

Aalto-yliopisto

Examples for Relaxed Reliability (1)

- Roles of nodes: Does everyone have to get everything?
  - Rather for group than for point-to-point communications
  - Some nodes may perform functions that require them to get all the data
  - Other nodes may drop out if they are not successful receiving everything
- Nodes may also be considered equal and just a quorum is needed
- For N communicating nodes, K-reliability means that only K out of N nodes need to receive the data
  - Useful and sufficient e.g. for replication
  - More difficult if the group attempts to obtain a coherent view
- …

Examples for Relaxed Reliability (2)

- Is all information equally important?
  - Is correctness of all information equally important?
  - Is timeliness of all information equally important?
- Unequal error protection
  - Protect certain pieces of information better than others
- Example 1: bits and bytes:
  - Provide a CRC and/or FEC only for parts of a packet (typically the beginning)
    - Allow less important parts of contents to contain bit errors (e.g., for audio)
    - But protect the parts essential for reproduction
  - Will result in lower frame loss rate, e.g., in wireless networks
- Example 2: packets
  - Provide FEC and/or retransmissions only for certain packets
    - The more essential part of the contents (e.g., video I frames, information changing rarely)
    - Accept losses for information that is updated frequently anyway or less important

Relaxed Reliability (3)

- How long is the information transmitted valid or useful?
  - Somewhat related to the soft state discussion
- Observation: once data is passed to the TCP layer, the data is doomed to be retransmitted until confirmed (or connection loss)
  - Regardless of whether the data is still useful at this point
  - Nice to have: allow to remove data again once no longer needed
    - Cross-layer interaction
- Example: meter readings
  - A complete log of readings (temperature, load, etc.) may be useful
  - But regular measurements (e.g., once every 100ms) will invalidate old data
    - Just transmit periodically; possibly support limited retransmissions
  - Yet capturing exceptional conditions may be important
    - So that this may be combined with more reliability depending on the values

Further Relaxations

- Sequencing
  - Reliability but no sequential delivery for all the data
  - Distinguishing multiple independently sequenced data streams
- Mixing reliable and unreliable transmission
- IETF: Stream Control Transmission Protocol
  - Origin: telephony signaling but now much more widespread applicability
- Congestion control without reliability
- IETF: Datagram Congestion Control Protocol (DCCP)

Discussion: Semantics of Reliability

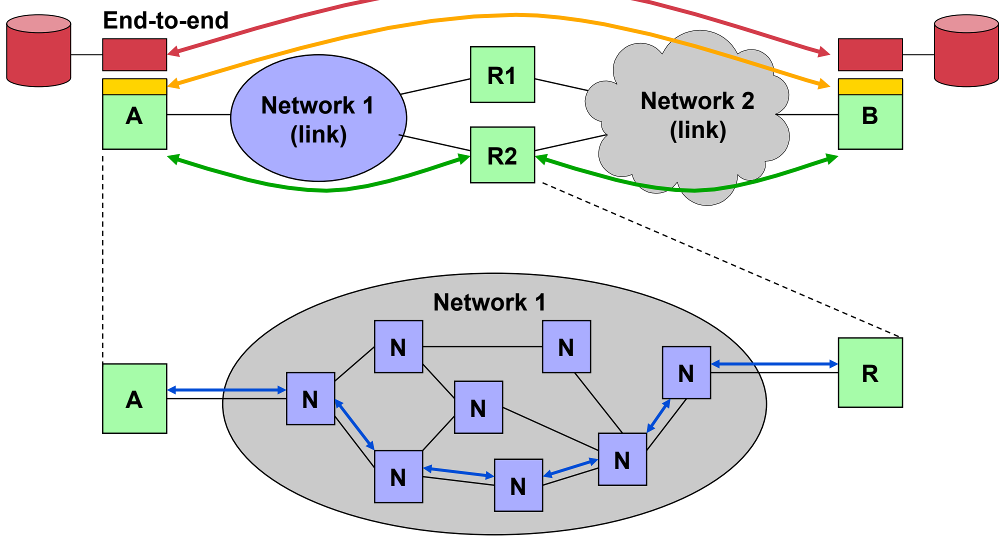
- Semantics of reliability ultimately depends on the application
- Hop-by-hop
  - Support by network elements on the path (such as routers)
    - Pro: More efficient retransmissions (not always all the time)
    - Cons: Routes may change, routers would spend resources (CPU, memory)
  - Support by intermediaries (hopefully) near the path (“overlays”)
    - Issues: Introduces additional points of failure, may cause suboptimal routing, …
  - Regardless of hop by hop support (optimization):
    the application is only interested in the end-to-end result of an operation
  - Beware of interacting control loops (hop-by-hop + end-to-end)
- End-to-end
  - Implementation exclusively on the end systems
  - Other elements may optimize but should not be able to have a negative impact
- What does end-to-end mean? (or: what is the *end*?)

Example: Careful File Transfer

- Move a file from a disk attached to machine A to a disk connected to machine B via some network
- Ensure complete and identical availability of the file on B's disk afterwards

- Proper reception, processing, and storage can only be assured by the application itself
  - It is the only entity aware of the real requirements
  - Needs to implement proper validation mechanisms anyway
- Transport and lower layer protocols can help performance
- The proper tradeoff requires careful thought!

Example: Careful File Transfer



Aalto-yliopisto

Low- vs. High-Level Implementation

- Lower layer implementation
  - ✓ May simplify applications or perform functions more efficiently
  - ✓ May be shared by numerous applications
  - But may be enforced on applications that do not need it
  - Operating on incomplete information may be less efficient
- Higher layer implementation
  - ✓ May be tailored to an application's needs
  - But may require the application (protocol) designer to deal with the issue
- Choice of several layers (network, transport, application)
- Trade-off is important!
  - Implies properly identifying "the ends"

Aalto-yliopisto

How much Reliability is needed?

- Again: Reliability semantics ultimately depend on the application
- Design and engineering tradeoff
  - Rely on existing transport protocols (TCP, more flexible now with SCTP)
    - Do not have to worry about getting the specification and the implementation right
    - Application protocol is often sufficient hassle already
    - Considerations on application-specific end-to-end reliability is required nevertheless
  - Do-it-yourself
    - Ultimate flexibility (and effort required)
    - Combine the mechanisms tailored to the application needs
    - Application Layer Framing (ALF)
      - Coined in the context of application-protocol-aware reliable multicast
- There is typically no single right solution