Protocols for dependable data transport in wireless sensor networks

Holger Karl
holger.karl@uni-paderborn.de
Andreas Willig (@ TU Berlin)
awillig@tkn.tu-berlin.de

Computer Networks Group
Universität Paderborn

Overview

- Dependability requirements
- Delivering single packets
- Delivering blocks of packets
- Delivering streams of packets

Dependability aspects

- ***Coverage & deployment***
  - Is there a sufficient number of nodes such that an event can be detected at all? Such that data can accurately measured?
  - How do they have to be deployed?
- ***Information accuracy***
  - Which of the measured data have to be transported where such that a desired accuracy is achieved?
  - How to deal with inaccurate measurements in the first place?
- ***Dependable data transport***
  - Once it is clear which data should arrive where, how to make sure that it actually arrives?
  - How to deal with ***transmission errors*** and ***omission errors/congestion***?

Focus of this tutorial

Dependability: Terminology

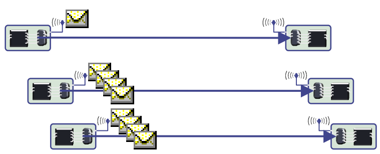
- "Dependable" is an umbrella term
- Main numerical metrics
  - ***(Steady state) availability*** – probability that a system is operational at any given point in time
    - Assumption: System can fail and will repair itself
  - ***Reliability at time t –*** Probability that system works correctly during the entire interval [0,t)
    - Assumption: It worked correctly at system start t=0
  - ***Responsiveness –*** Probability of meeting a deadline
    - Even in presence of some – to be defined – faults
  - ***Packet success probability*** – Probability that a packet (correctly) reaches its destination
    - Related: packet error rate, packet loss rate
  - ***Bit error rate –*** Probability of an incorrect bit
    - Channel model determines precise error patterns

Dependability constraints

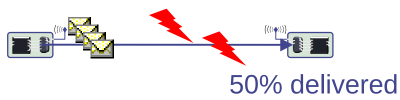
- Wireless sensor networks (WSN) have unique constraints for dependable data delivery
  - Transmission errors over a wireless channel
  - Limited computational resources in a WSN node
  - Limited memory
  - Limited time (deadlines)
  - Limited dependability of individual nodes
- Standard mechanisms: Redundancy
  - Redundancy in nodes, transmission
  - Forward and backward error recovery
  - Combinations are necessary!

Dependable data transport – context

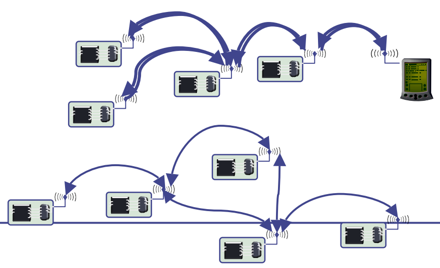
- Items to be delivered
  - Single packet
  - Block of packets
  - Stream of packets



- Level of guarantee
  - Guaranteed delivery
  - Stochastic delivery



- Involved entities
  - Sensor(s) to sink
  - Sink to sensors
  - Sensors to sensors



Constraints

- Energy
  - Send as few packets as possible
  - Send with low power $\rightarrow$ high error rates
  - Avoid retransmissions
  - Short packets $\rightarrow$ 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

Overview

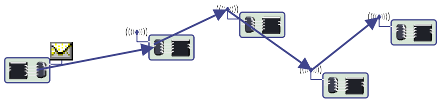
- Dependability requirements
- ***Delivering single packets***
  - Single path
  - Multiple paths
  - Gossiping-based approaches
  - Multiple receivers
- Delivering blocks of packets
- Delivering streams of packets

Delivering single packets – main options

- 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 to single receiver over single path

- Single, multi-hop path is giving by some routing protocol



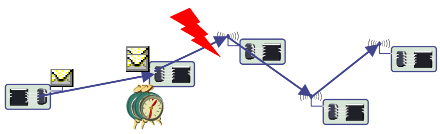
- Issues: Which node
  - Detects losses (using which indicators)?
  - Requests retransmissions?
  - Carries out retransmissions?

Detecting & signaling losses in single packet delivery

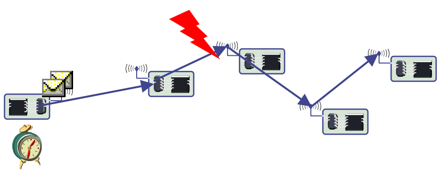
- Detecting loss of a *single packet*:
  Only positive acknowledgements (ACK) feasible
  - Negative acks (NACK) not an option – receiver usually does not know a packet should have arrived, has no incentive to send a NACK
- Which node sends ACKs (avoiding retransmissions)?
  - At each intermediate node, 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

Carrying out retransmissions

- For link layer acknowledgements: Neighboring node



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

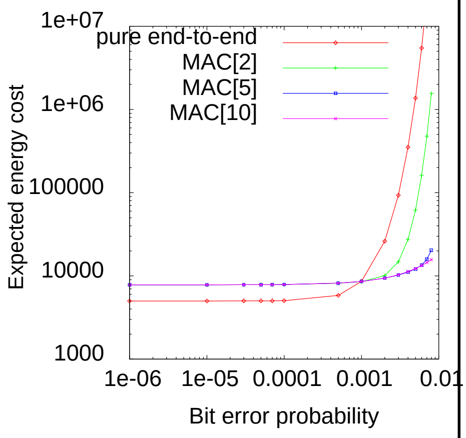


Question: Could an intermediate node help in an end-to-end scheme? How to detect need for retransmissions? How to retransmit?

Tradeoff: End-to-end vs. link-layer retransmission

- Scenario: Single packet, n hops from source to destination, BSC channel
- Transport-layer, end-to-end retransmission: Always
- Link-layer retransmissions: Vary number of maximum attempts
  - Drop packet if not successful within that limit

$\rightarrow$ For good channels, use end-to-end scheme; else local retransmit

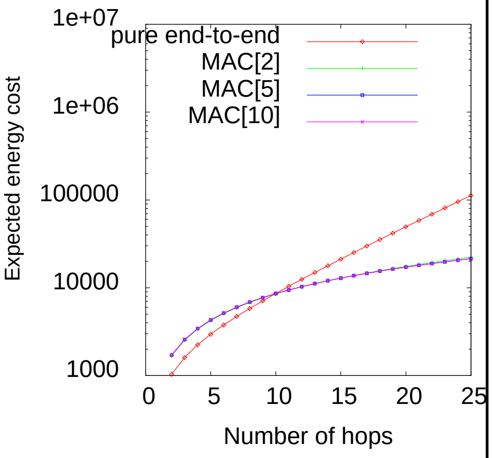


Tradeoff: End-to-end vs. link-layer retransmission

- Same scenario, varying number of hops
  - BER=0.001 of BSC channel fixed

$\rightarrow$ Use link-layer retransmissions only for longer routes

In both figures, difference between maximum link-layer retries schemes is small. Why?



Example schemes: HHR and HHRA

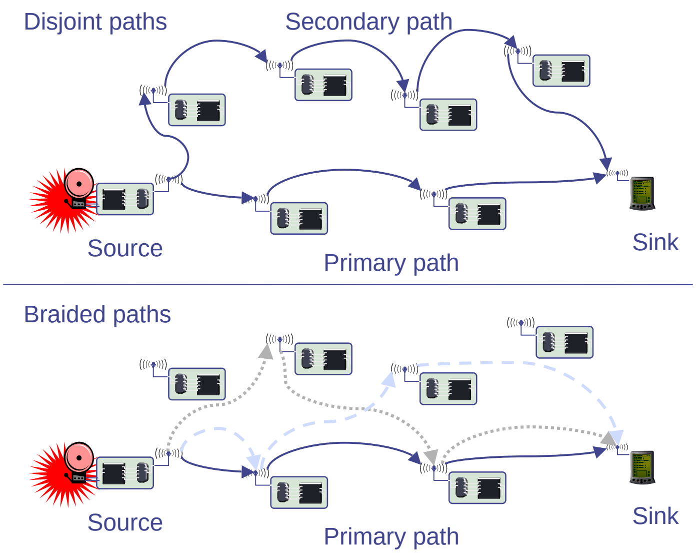
- ***Hop-by-hop reliability (HHR)***
  - Idea: Locally improve probability of packet transmission, but do not use packet retransmission
  - Instead, simply repeat packet a few times – a repetition code
  - Choose number of repetitions per node such that resulting end-to-end delivery probability matches requirements
- ***Hop-by-hop reliability with Acknowledgements (HHRA)***
  - Node sends a number of packets, but pauses after each packet to wait for acknowledgement
  - If received, abort further packet transmissions

What happens in bursty channels?

Multiple paths

- Types of : disjoint or braided
- Usage: default and alternative routes
- Usage: simultaneous
  - Send same packet
  - Send redundant fragments
- Example: ReInForM

Multiple paths: Disjoint or braided



Using multiple paths

- Alternating use
  - Send packet over the currently “selected” path
  - If path breaks, select alternative path
  - Or/and: repair original path locally
- Simultaneous use
  - Send the complete packet over some or all of the multiple paths simultaneously
  - Send packet fragments over several paths
    - But endow fragments with redundancy
    - Only some fragments suffice to reconstruct original packet

Example: ReInForM

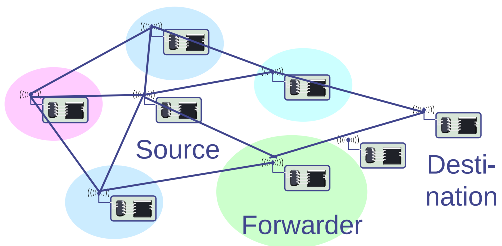
- Goal: Send packet over multiple paths to meet a delivery probability *P*
- Assumptions:
  - Independent paths, BSC
  - Nodes know their "local" packet error rate *e*
- Step 1: Source node decides how many paths to use
  - Success probability over a *single* path with n_s hops: $1-(1-e)^{n_s}$
  - Success probability over *P* paths: $1-(1-(1-e)^{n_s})^P$
  - Should be $\geq r_s$, solve for P:

$$P = \frac{\log(1 - r_s)}{\log\left(1 - (1 - e)^{n_s}\right)}$$

Note there is no floor/ceiling in this formula

ReInForM – Forwarding to neighbors

- Source node picks a ***forwarder*** closer to destination than itself
- Remaining neighbors: $P' = P - (1-e_s)$
- Choose P´ neighbors to additionally forward packet
  - If possible, only neighbors closer to destination
  - If not sufficient, use neighbors same hop distance
  - If not sufficient, use further away neighbors

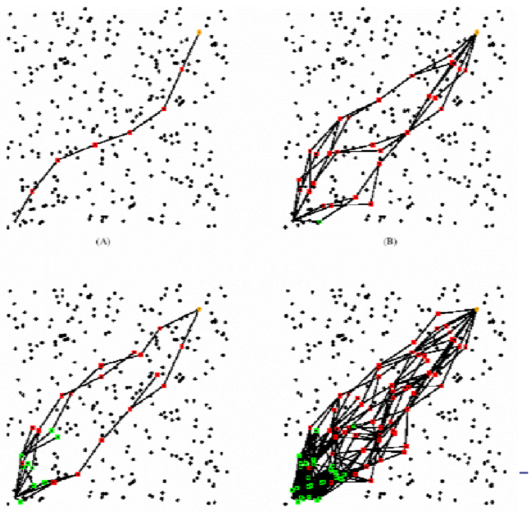


- Packet contains
  - Source & destination
  - Forwarder identity
  - Source packet error rate
  - Number of paths each neighbor should construct

ReInForM – Behavior of neighbors

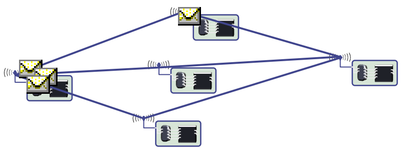
- Forwarder behaves just like a source
- Non-primary forwarders locally compute over how many paths they are supposed to forward the packet
  - If number of paths < 1, node only forwards with according probability

ReInForM load-balancing behavior for multiple packet transmissions



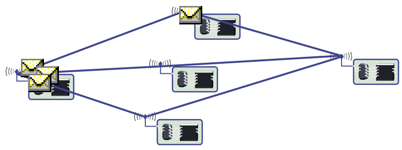
Gossiping-based approaches

- What to do when not routes are available?
  - Flooding – all nodes rebroadcast a received packet – not efficient
  - ***Gossiping*** – only *some* nodes rebroadcast?



- Problem: Which node rebroadcasts?
  - Deterministic choice (e.g., backbone construction): Overhead
  - Random choice: Forwarding probability?
- Gossiping is greatly helped by direction to destination!

Forwarding probability for gossiping



- Assumption: All nodes know
  - destination to direction,
  - number of neighbors *k*,
  - packet error probability *e*
- Goal: On average, a single node should forward packet
- Expected number of packets received: k(1-e)
- Each node receiving a packet forwards with probability
  $$P_{forward} = 1/k(1-e)$$
  - Packet needs to contain *k*, *e*
- Problem: Gossip might die out

Flooding based on neighborhood behavior

- Suppose a packet should be distributed to all nodes
- Suppose a node can observe behavior of its neighbors
- When to actually forward a new packet?
  - Immediately? All nodes will then forward, some needlessly
  - Wait and check neighbors? When many neighbors have already forwarded the packet, is it worthwhile to do so as well?
- Observation (for uniformly distributed networks):
  - When $k \geq 4$ neighbors have already forwarded a packet, the additional coverage gained by forwarding it one more time is $\leq 0.05\%$

  $\rightarrow$ Wait random time, count neighbors' forwards, only forward when not already done so in neighborhood

Multiple receivers

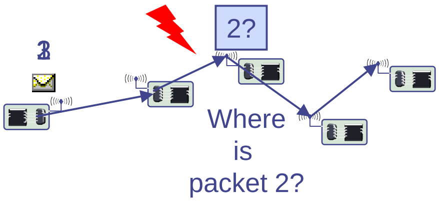
- Deliver a single packet to multiple receivers: Multicast
  - Formally: Steiner tree problem, NP complete
  - Constructing Steiner tree for a single packet probably excessive; might pay off for multiple packets
- Problem: ACK implosion
  - Many receivers send ACKs to a single source
  - Source/nodes near source are overloaded
- Combination with ACK aggregation

Overview

- Dependability requirements
- Delivering single packets
- ***Delivering blocks of packets***
  - Opportunity: Caching in intermediate nodes
  - Example: Pump Slowly, Fetch Quickly (PSFQ)
  - Example: Reliable Multisegment Transport (RMST)
- Delivering streams of packets

Delivering blocks of packets

- 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



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

Example: Pump Slowly Fetch Quickly (PSFQ)

- Goal: Distribute block of packets to from one sender to multiple receivers (sink to sensors)
  - E.g., code update → losses are not tolerable, delay not critical
  - Routing structure is assumed to be known
- Basic operation
  - Source ***pumps*** data into network
    - Using broadcast, *large inter-packet gap time*
  - Intermediate nodes store packets, forward if in-sequence
  - Out-of-sequence: buffer, request missing packet(s) – ***fetch*** operation (a NACK)
    - Previous node resends missing packet → local recovery
    - Assumption: packet is available ← no congestion, only channel errors

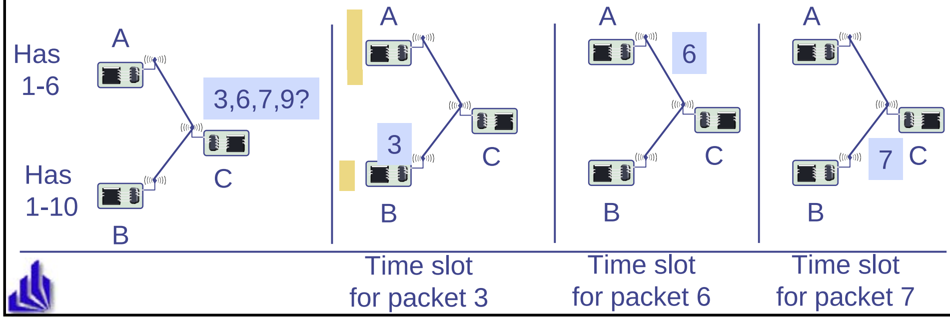
  → Pumping is slow, fetching is quick

PSFQ protocol details

- How big an inter-packet gap?
  - Big enough to accommodate at least one, better several fetch operations
  - Probability that next packet arrives when the previous one has not yet been repaired should be small
- When to forward an in-sequence packet?
  - Wait random time, only forward when $\leq$ 3 neighbors have forwarded
- Handle out-of-order packet?
  - Do not forward, fill the gap first by fetching $\rightarrow$ avoid loss propagation

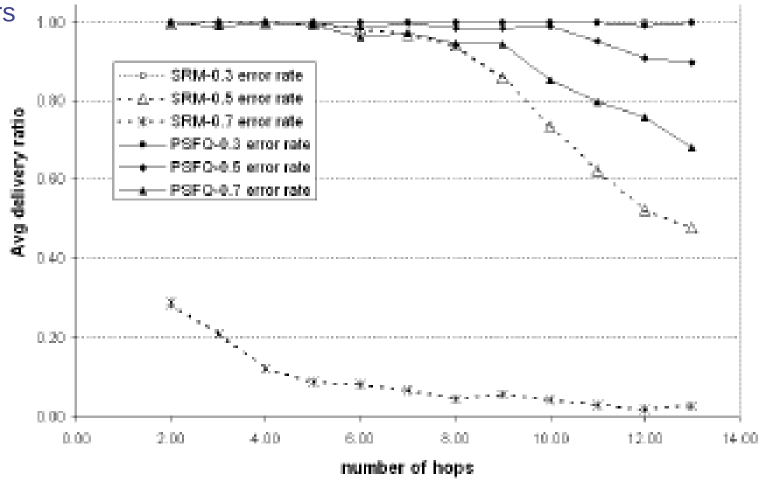
PSFQ protocol details (2)

- How to handle fetch requests (NACKs)?
  - Fetch request are broadcast, might arrive at multiple nodes
  - Nodes receiving NACK might themselves not have all requested packets
  - Use a slotted resend mechanism for requested packets – each one corresponds to a time slot, filled by node if requested packet available
  - Example: Node C requests 3,6,7,9 in NACK



PSFQ performance: Comparison with multicast

- Comparison case: Scalable Reliable Mutlicast (SRM)
  - Provides similar service
  - Main difference: in-sequence not enforced, end-of-block treatment differs
- PSFQ works up to higher error rates



Reliable Multisegment Transport (RMST)

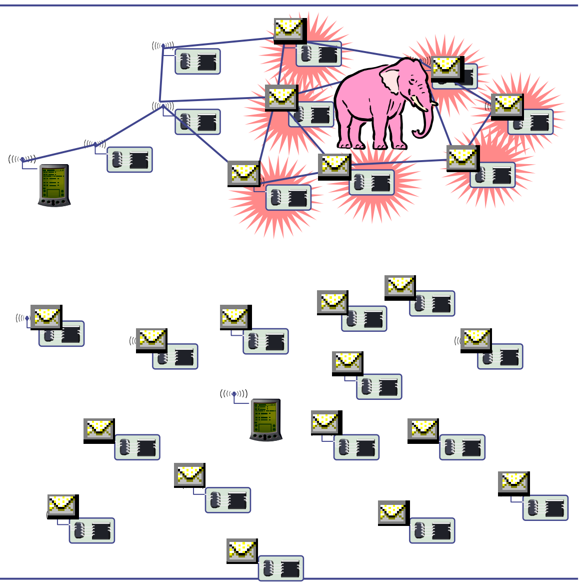
- Goal: Dependable delivery of large data blocks from multiple sensors to a single sink
  - Data block is fragmented – collect all fragments, deliver to sink
  - Tightly coupled with directed diffusion
  - Does not include congestion control, time bounds
- Basic RMST mechanisms
  - MAC-layer retransmissions (802.11, full procedure: RTS/CTS, …)
  - RMST caches fragments, checks for missing fragments
  - When gap is detected NACKs are sent back towards the sources
  - NACKs are served by intermediate node if fragment is present
  - Else: NACK forwarded, but only rarely – e.g., when path has not changed
  - To catch remaining errors, sources occasionally retransmit all

Overview

- Dependability requirements
- Delivering single packets
- Delivering blocks of packets
- ***Delivering streams of packets***
    - Additional opportunity: Control rate
    - Control rate of individual nodes: ESRT
    - Control number of active nodes: Gur game

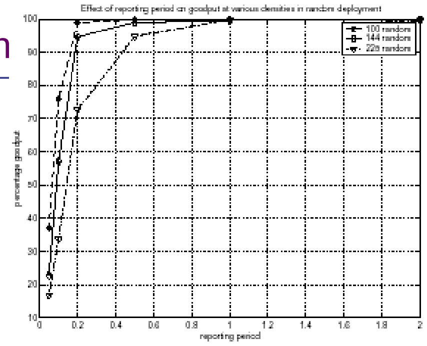
Streams of packets may lead to congestion

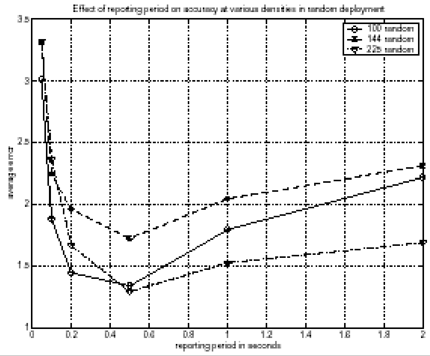
- When several sensors observe an event and try to periodically report it, congestion around event may set it
- When many sensors stream data to a sink, congestion around the sink may occur



Consequences of congestion

- Congestion can have surprising consequences
- More frequently reporting readings can reduce goodput and accuracy
  - Owing to increased packet loss
- Using more nodes can reduce network lifetime





Detecting congestion

- TCP: Detect congestion by missing acknowledgements
  - Here not applicable if no ACKs are used
- ***Locally*** detect congestion
  - Intuition: Node is congested if its buffer fills up
  - Rule: “Congested = buffer level above threshold” is overly simplistic
  - Need to take growth rate into account as well
    - Occupancy not a good indicator when packets can be lost in the channel
- Problematic: Interaction with MAC
  - CSMA-type MACs: high channel utilization = congestion; easy to detect
  - TDMA-type MACs: high channel utilization not problematic for throughput; congestion more difficult to detect

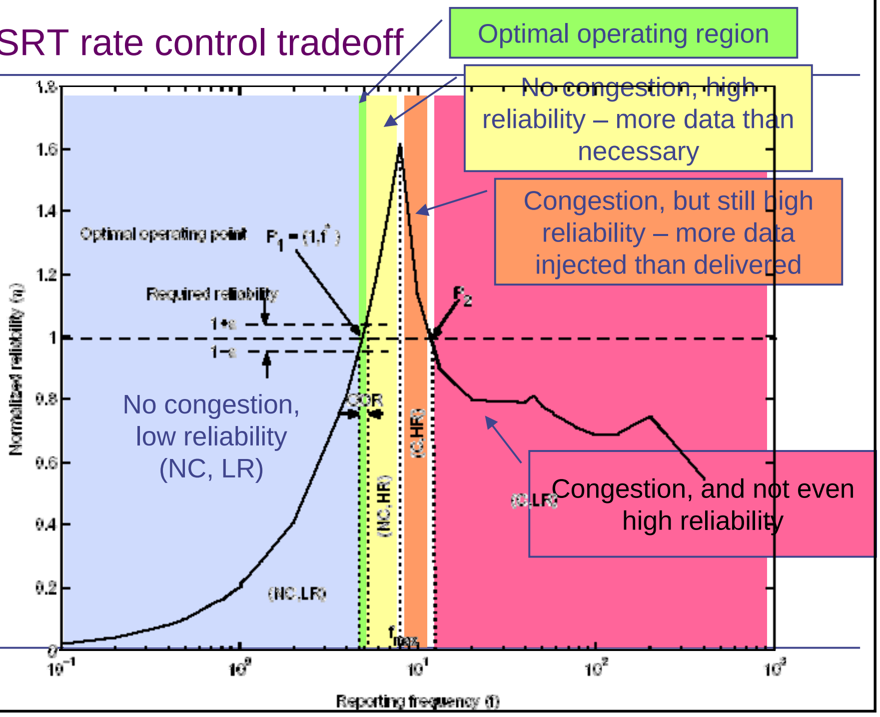
Congestion handling

- Once congestion is (locally) detected, how to handle it?
- Option 1: Drop packets
  - No alternative anyways when buffers overflow
  - Drop tail, random (early) drop (for TCP), …
  - Better: drop semantically less important packet
- Option 2: Control sending rate of individual node
  - Rate of locally generated packets
  - Rate of remote packets to be forwarded $\rightarrow$ ***backpressure***
- Option 3: Control how many nodes are sending
- Option 4: Aggregation, in-network processing

Rate control: Event-to-Sink Reliable Transport (ESRT)

- Situation: Multiple sensors periodically report to sink
  - Sink needs sufficient number of packets, from *any* source
- Control knob: control sensors' reporting rate f_i
  - Ensure: per *decision period* τ, +/- R packets are delivered
  - Formally: r_i packets actually received in period i,
  - Target: $\eta_i = r_i/R \in [1-\varepsilon, 1+\varepsilon]$
- Sink computes f_{i+1} based on f_i, η_i
  - Broadcasts to all sources directly (high power)

ESRT rate control tradeoff



ESRT's adaptation of source frequencies

- No congestion, low reliability: Increase data rate
  - $f_{i+1} = f_i / \eta_i$
  - Note: $\eta_i < 1$ here (less data arrives than necessary)
- Optimal operating region: do nothing
- No congestion, high reliability: moderate reduction of sending rate useful
  - $f_{i+1} = f_i/2 \, (1 + 1/\eta_i)$
- Congestion, high reliability: quicker reduction of rate
  - $f_{i+1} = f_i / \eta_i$
  - Note: $\eta_i > 1$ here (more data arrives than necessary)
- Congestion, low reliability: even quicker reduction of rate
  - $f_{i+1} = f_i^{\eta_i/k}$
  - k: number of consecutive rounds in this state

Control how many nodes are sending

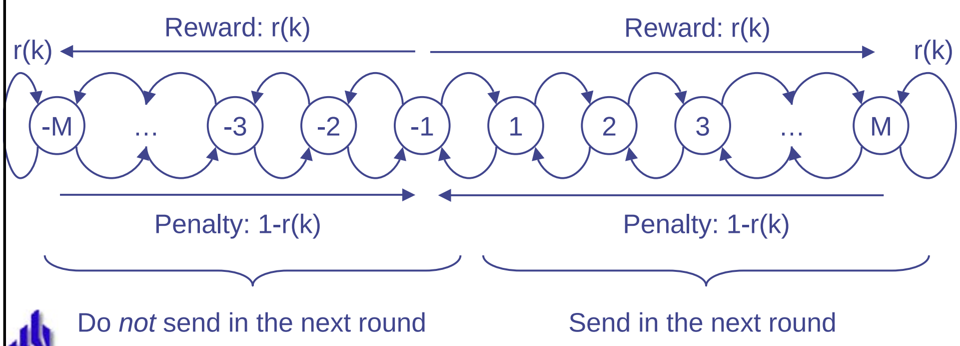
- Scenario: Nodes send at a given rate, cannot be controlled
- Option: Turn on or off nodes to avoid congestion, achieve desired target number of packets k* per round
  - If *total* number of nodes N known, easy: Simply send probability k*/N to all nodes; each node sends with this probability

- What to do if number of nodes N not known?

$\rightarrow$ ***Gur game***

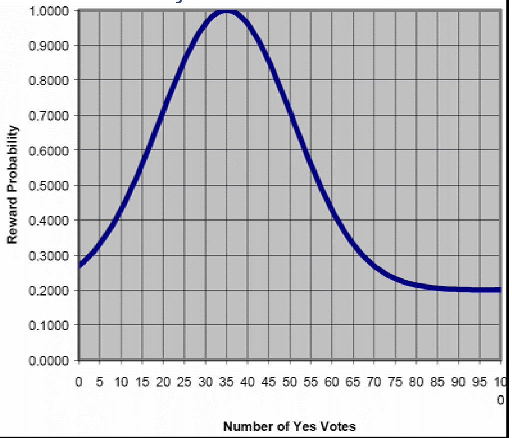
Gur game

- N nodes, unaware of each other; 1 referee
- Referee, in each round:
  - Counts number k of packets (assumption: no packet loss)
  - Determines ***reward probability r(k)***, sends r(k) to all nodes
- Each player: rewards itself with probability r(k), penalizes with probability 1-r(k)
  - Rewards/penalties: Moves in finite state machine



Gur game: How to choose r(k)?

- Intuition
  - When received number of packets k is close to k*, the right number of nodes are sending
  - Thus, the right mixture of send/not send states is present
  - → Nodes should stay on the side where they are
  - → Rewards should be high
- Formally
  - Reward function is maximal at k*
  - Example: See figure



Conclusion

- Transport protocols have considerable impact on the service rendered by a wireless sensor networks
- Various facets – no "one size fits all" solution in sight
- Still a relatively unexplored areas

- Items not covered
  - Relation to coverage issues
  - TCP in WSN? Gateways?
  - Aggregation? In-network processing?