(Looking for) A SWiss Army Knife for Wireless Sensor Networks

Ioanis Nikolaidis

University of Alberta

Computing Science Department

<yannis@cs.ualberta.ca>

UNIVERSITY OF ALBERTA

The Cast

- Ioanis Nikolaidis
- Pawel Gburzynski
- Mario Nascimento
- Eleni Stroulia
- Wlodek Olesinski
- Baljeet Malhotra
- Nicholas Boers
- Ashikur Rahman
- Benyamin Shimony

Motivation

- Distance between literature and practice.
- Questions about the proper development tools.
- Search for what might be useful abstractions.
- Ability to quickly move from idea to realization.

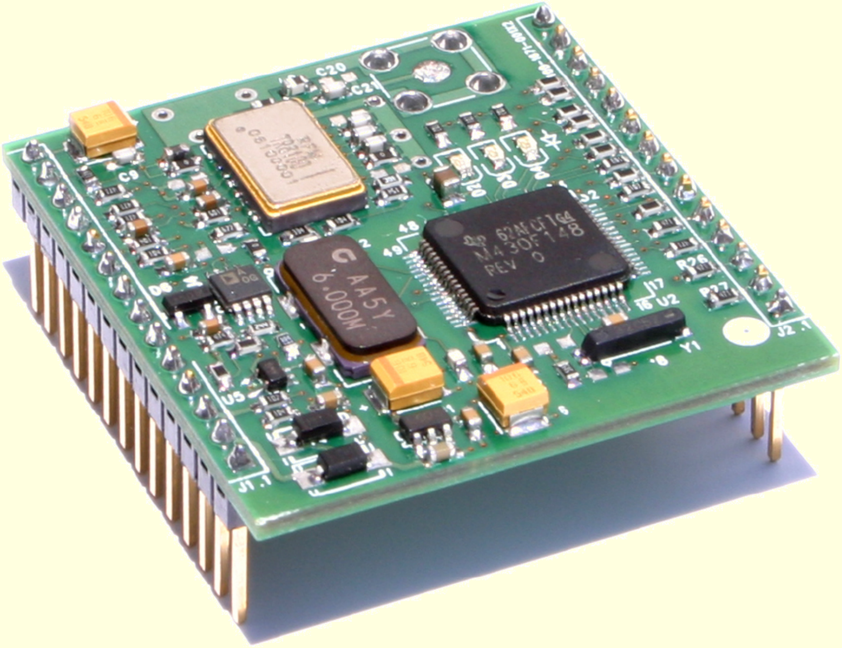
The Main Problem

- A significant part of the literature is implicitly assuming high capability devices (even for “elementary tasks” like routing).
  - Will they work in “really small” platforms?
- Moore's Law can be interpreted in two ways. The persistence on a single interpretation hinders our appreciation of future possibilities.
- What is more “important”?
  - 10 billion nodes at $10 a piece?
  - 200 million nodes at $500 a piece?
- What is easier to “upgrade”?
  - An electric shaver?
  - Your most recent version of MS Windows?

Our Toys

Platform: DM2200

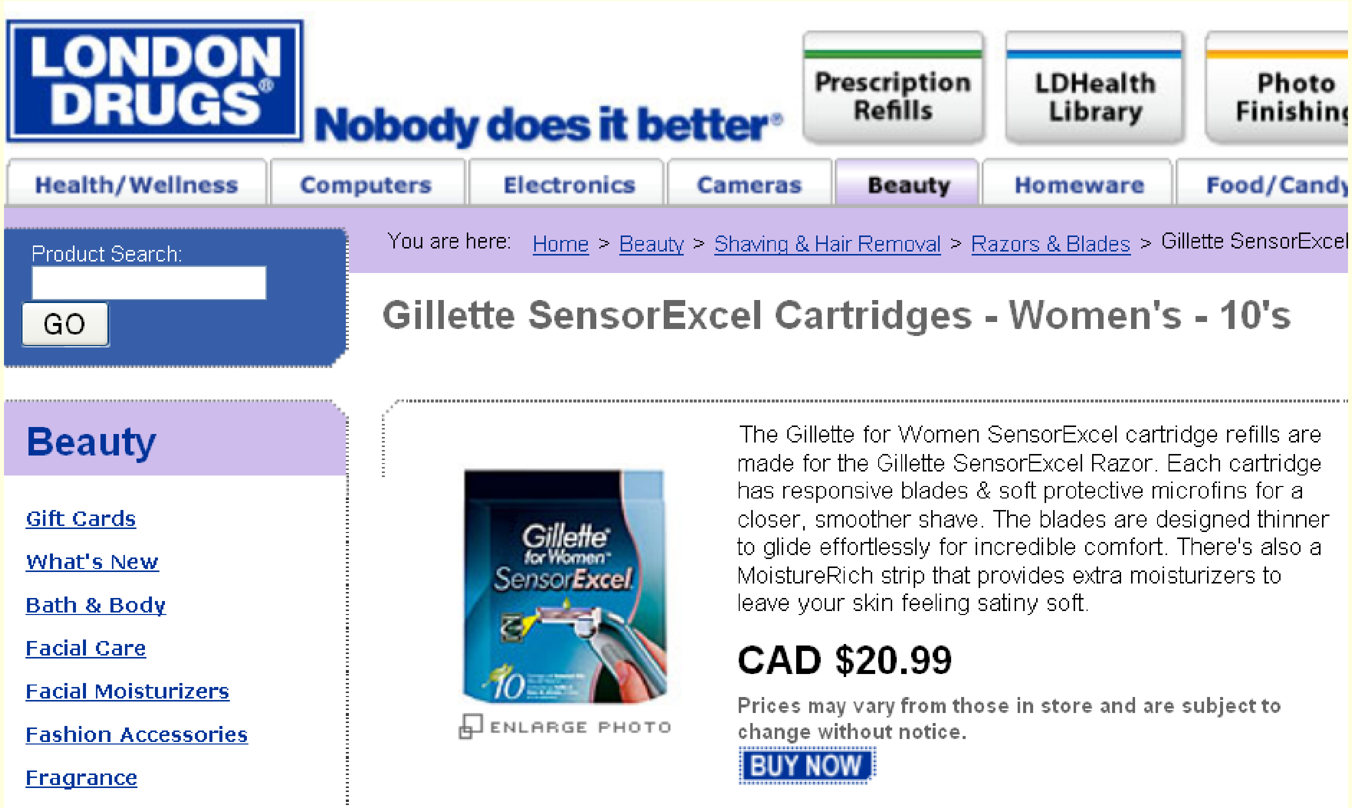
- RFM TR8100
- TI MSP430F148
  - 48 KB Flash
  - 2 KB RAM
- 916.5 MHz
  - 916.3-916.7
  - OOK on BPSK spreading
  - 9.6 kbps



www.rfm.com

Disposable Computing

- Devices $20 or less can be thrown away after a (possibly short) useful life, but still need code.



The Paradox

- Cheap devices require expensive development!

- One has to account for:
  - Code development cost.
  - Code reuse capabilities.

- Code production is the bottleneck to testing the great ideas found in the literature.
  - Simulation is a poor substitute.

- What helps development:
  - Sufficiently high-level abstractions (but limited OS).
  - A “natural” composition mechanism.

A Word about Standardization

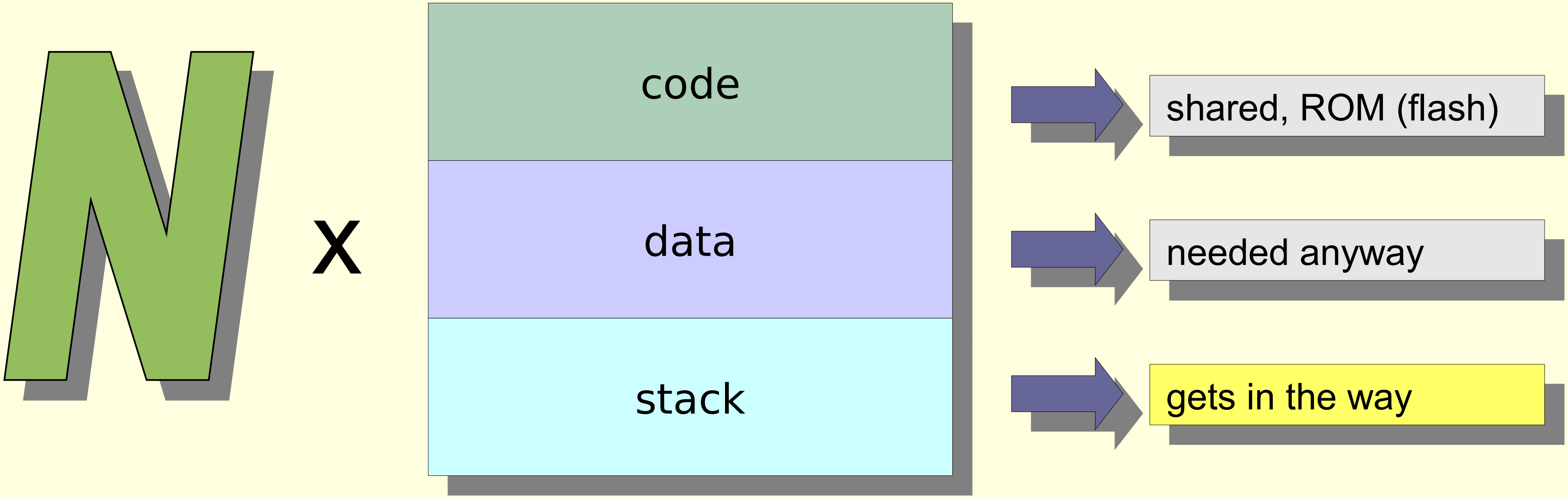
- Claim: "*Wireless Sensor Networks are not yet successful because the protocols have only recently been standardized, e.g., ZigBee.*"
- What we should be asking is:
  - "*Does the developer spend more time because of DL+PHY lack of standardization?*"
  - "*Does the developer's work become significantly more difficult when dealing with a proprietary DL+PHY vs. a standard one?*"
- Standardization at the higher layers is a struggle.
  - Some brave efforts from the Open Geospatial Consortium are reasons for hope, albeit "verbose".

The TinyOS Story

- Admittedly the first serious attempt to provide an open-source OS for wireless sensor networks.
- Currently, a source of frustration for many developers. Value added products are not “free”.

- Model: event handlers and tasks
  - Event handlers cannot be preempted.
  - No task preemption as such (in “vanilla” TinyOS).
  - Tasks executed in order posted (in “vanilla” TinyOS).
  - No multi-threading as such.
  - Dynamic memory allocation curtailed.
  - “Wired” components useful but potentially hard to track down how overall functionality composed.

The Essence of the Problem

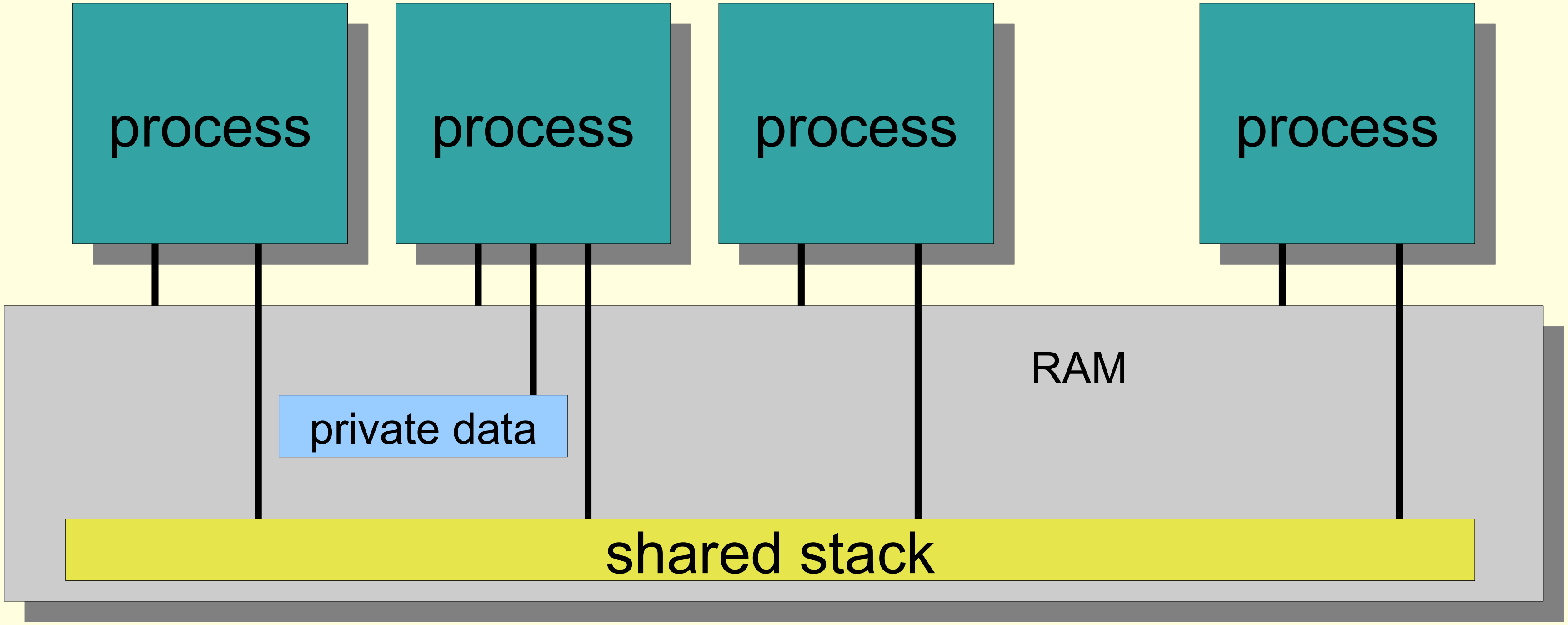


The PicOS Alternative

- Protocol designers can (ought to be able to!) describe protocols as finite state machines.
- A thread model allows for natural expression of concurrency across “independent” strands of logic.

- PicOS: an OS tuned to small platforms:
  - Implement concurrency as co-routines.
  - Co-routines reduce the stack requirements.
  - Express each process/thread as a FSM.
  - Process preemption possible at state boundaries.
  - Interrupts can preempt processes.
  - Interrupts deliver “events” to processes/threads.

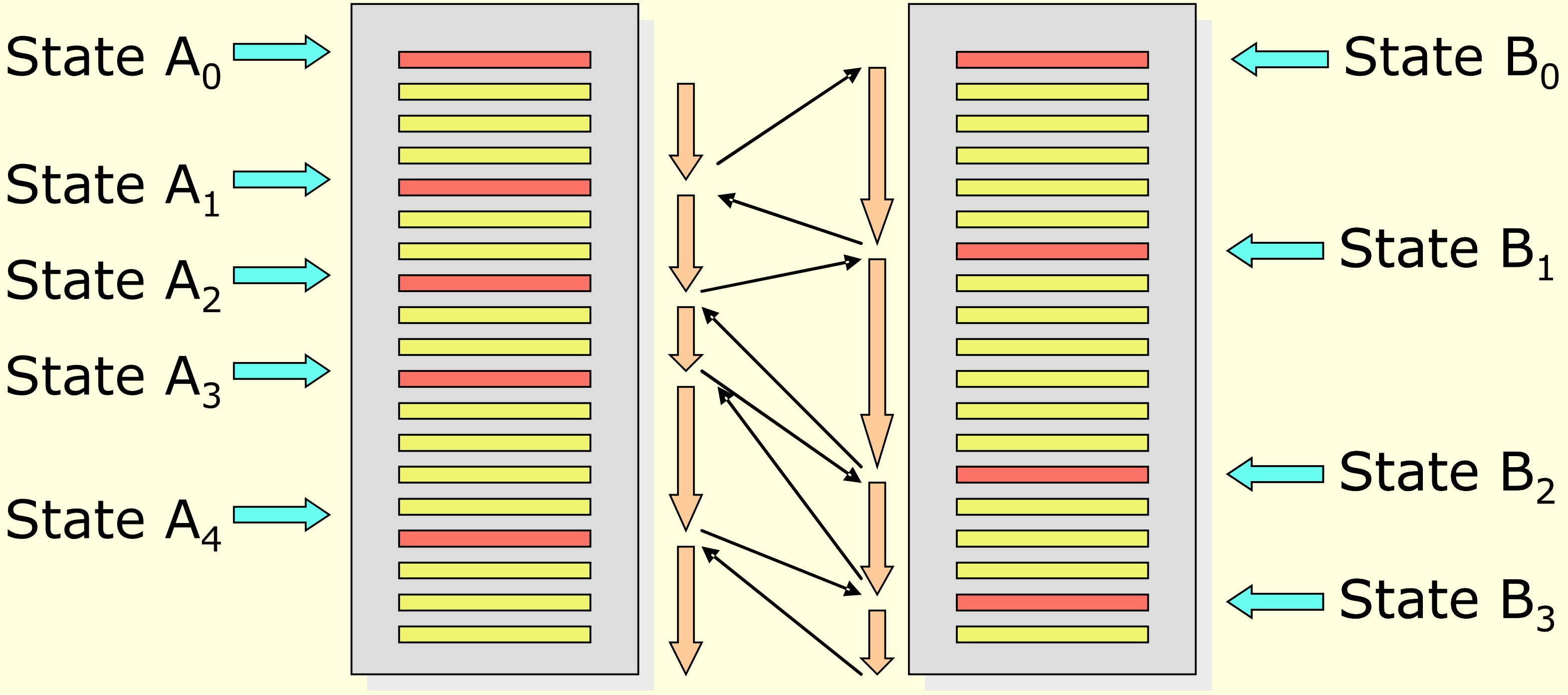
The PicOS Solution



20 bytes of RAM per process

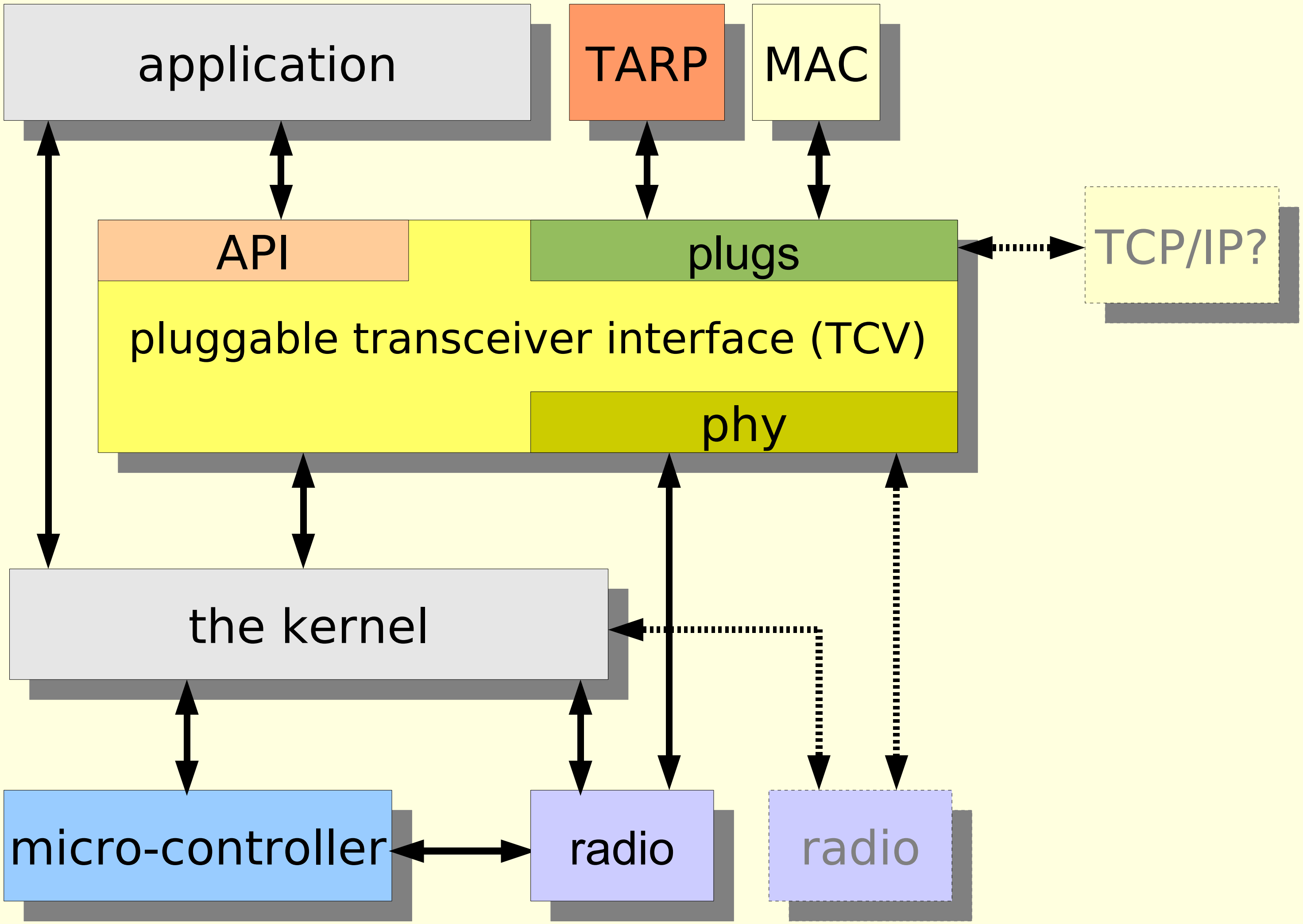
64 bytes of (global) stack goes a long way

Multi-Threading and CoRoutines



```
process (sniffer, sess_t)
    char c;
    entry (RC_TRY)
        data->packet = tcv_rnp (RC_TRY, efd);
        data->length = tcv_left (data->packet);
    entry (RC_PASS)
        if (data->user != US_READY) {
            wait (&data->user, RC_PASS);
            delay (1000, RC_LOCKED);
            release;
        }
        ...
    entry (RC_LOCKED)
        ...
    entry (RC_ENP)
        tcv_endp (data->packet);
        signal (&data->packet);
        proceed (RC_TRY);
endprocess (1)
```

Architecture



A Higher Level View

- Building the OS is fine, but what about some basic abstractions to help the developer? Which ones?

- “Solution”: read current literature, find patterns and translate them to a handful of abstractions.

- Some first attempts:

  The Emergence of Networking Abstractions and Techniques in TinyOS, by Levis et al. (NSDI 2004).

  Logical Neighborhoods : A Programming Abstraction for Wireless Sensor Networks, by Luca & Gian (DCOSS 2006).

Prime Candidates for Abstraction

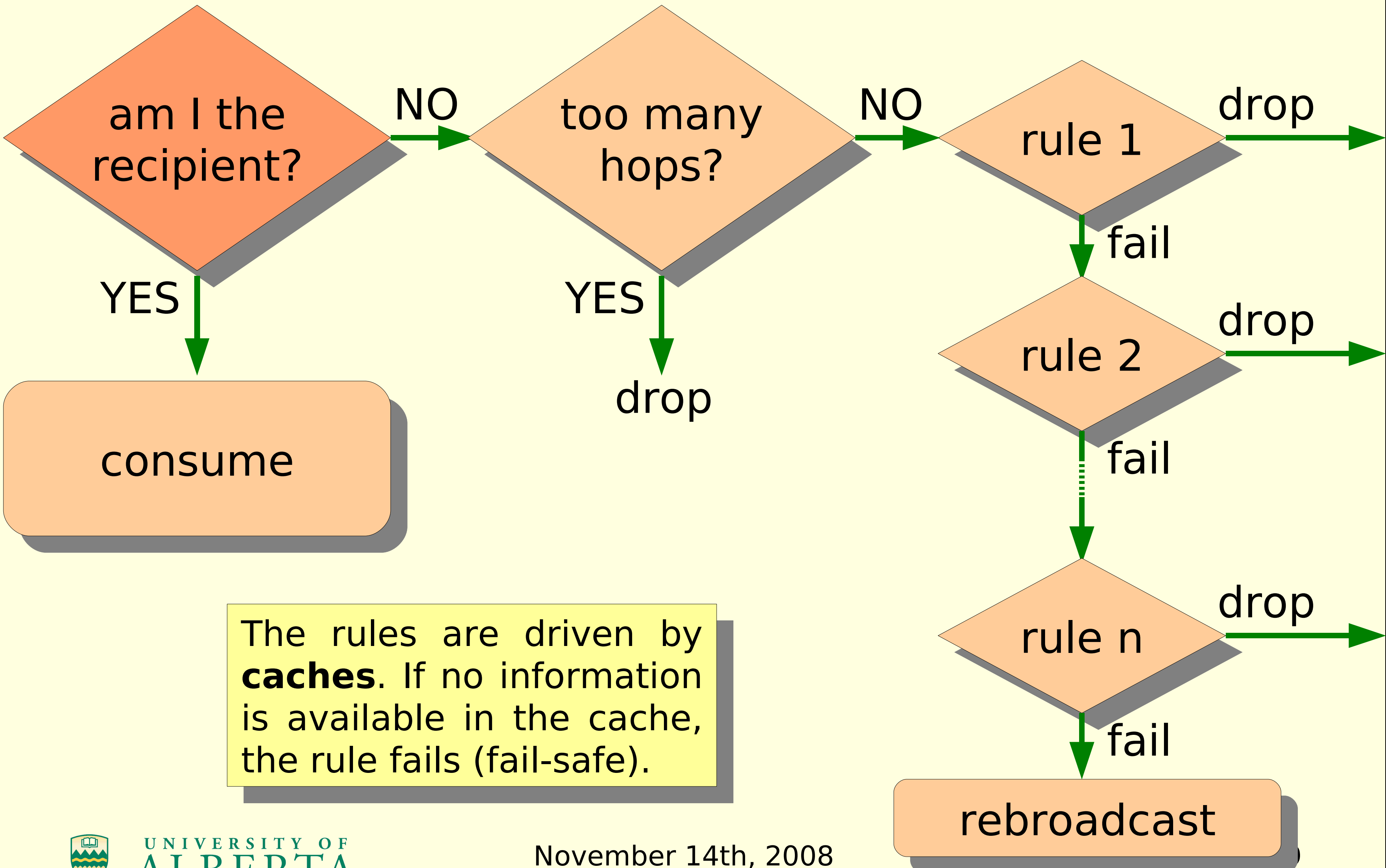
- Path (route)
- Neighborhood
- Spanning structure
- Region
- Duty cycle
- ...

Paths

- Useful for all the obvious reasons (getting from A to B, **without** concerns of how to get there).
  - Why do we then care so much about the "next hop" and the maintenance of such information?
  - Most of the literature will have you believe we should. It need not be so.
  - Moreover, there is no reason for the intermediate nodes to even be identified as "next hop"!

- We should care about a "path" which is customizable to fit arbitrary forwarding decisions.
  - Trade node capabilities for routing performance.
  - TARP: Tiny Ad-Hoc Routing Protocol

How to Route in Wireless Sensors



The rules are driven by **caches**. If no information is available in the cache, the rule fails (fail-safe).

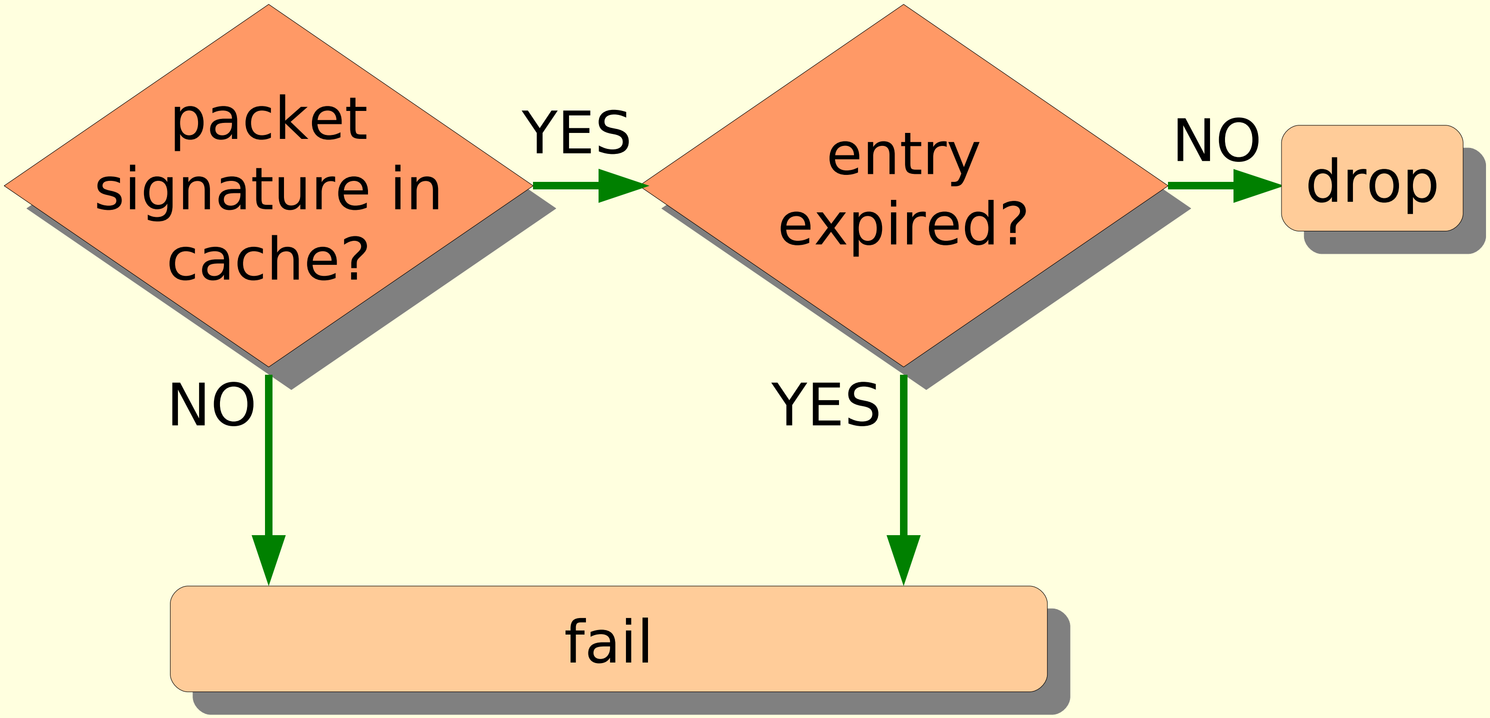
TARP Header Information

Packet "Signature"

D	destination	
S	source	
s	session tag	4
n	packet number	5
k	version (retransmission count)	4
r	hop number limit	5
h_f	hops so far	5
h_b	total # of hops on reverse path	5
m	slack	3
opf	optimal path flag	1

(example length in bits)

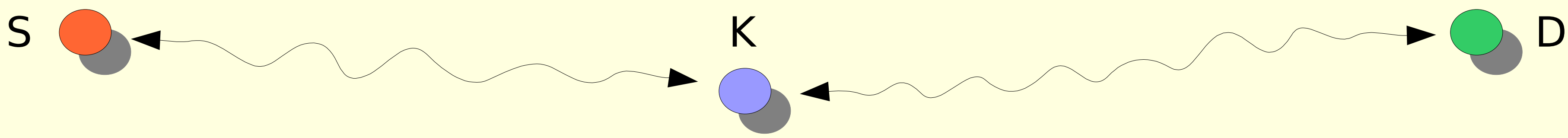
Expressiveness: Drop Duplicates



Cache key/entries:
<D,S,s,n,k>

The cache replacement policy is under the programmer's control. (Example: expiration time proportional to expected distance to destination.)

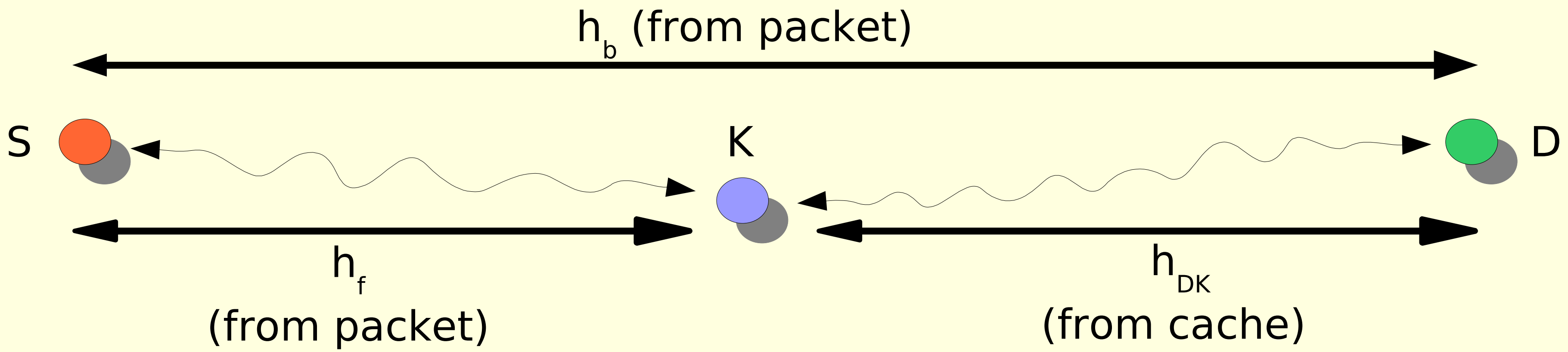
Expressiveness: Suboptimal Path Discard



Cache key:
<N>
Cache entries:
<N,h_{NK},C_{NK}>

For each arriving packet from N, h_f, is stored as the h_{NK} value.
(Since duplicates already discarded, h_{NK} tends to represent the shortest path from N to K.)

SPD (cont'd)



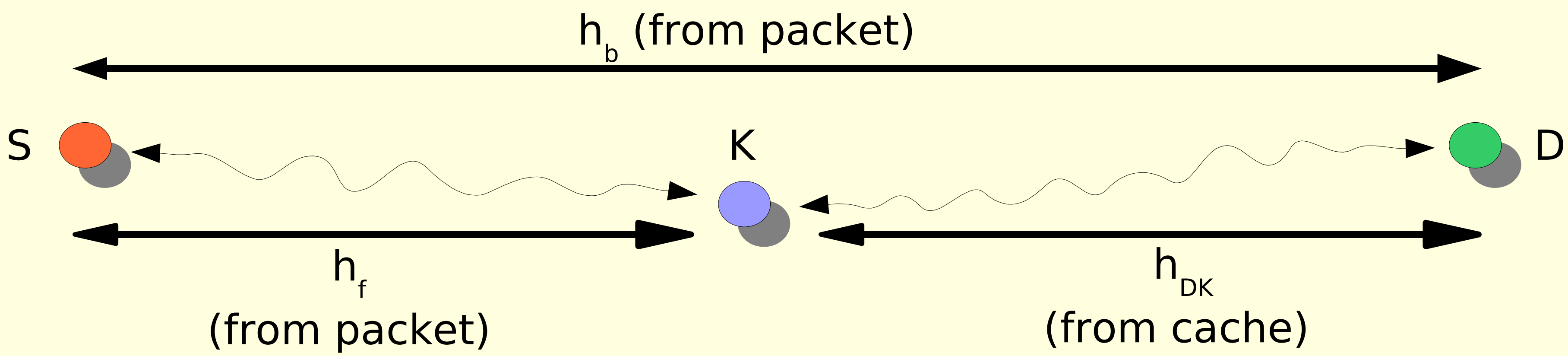
Suppose a packet from S arrives with a particular h_b, node K checks whether

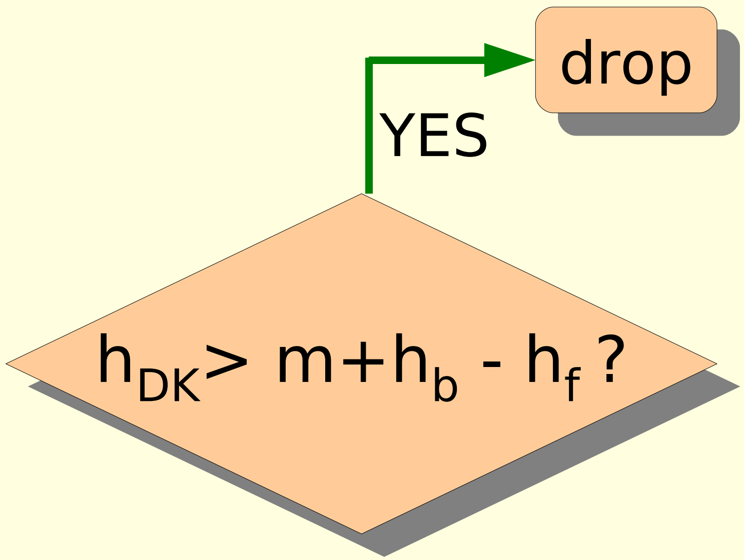
$$h_{DK} > h_b - h_f$$

if yes, then it is very likely that a better path exists from S to D not going through K. It looks like a good idea to drop the packet.

Except the paths may not be symmetric, and we need to be able to recover from node failures and/or mobility.

Relaxing SPD

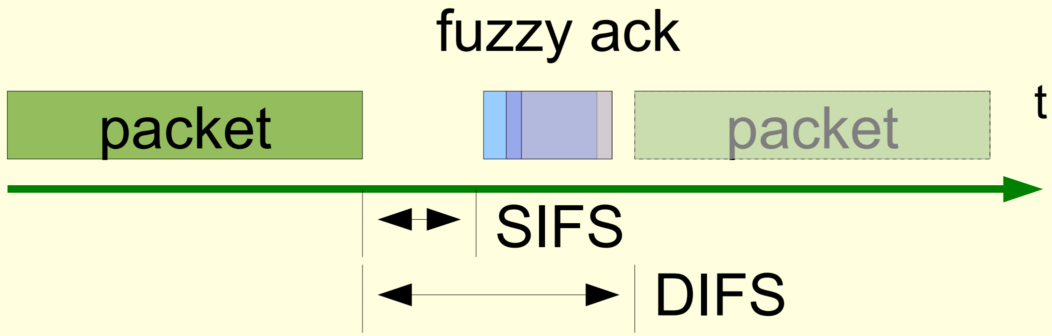




Increment C_{NK} every time the rule is successful. When it reaches a certain threshold, it forcibly fails and resets C_{NK}.

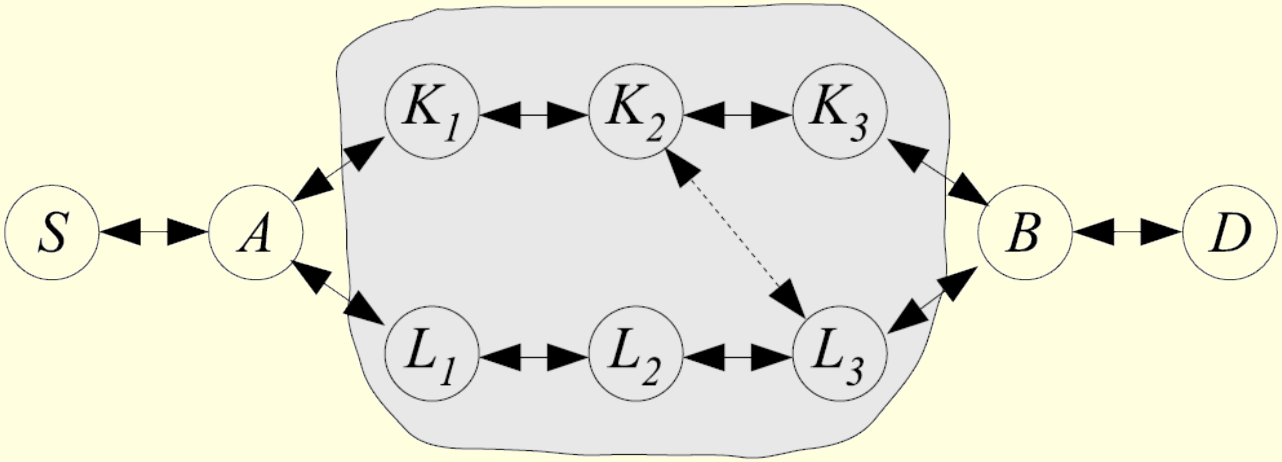
Additional Considerations

- Short burst of activity (fuzzy ack) to act as an implicit confirmation that the packet will not be dropped. Not a proper frame, but akin to a “tone”.



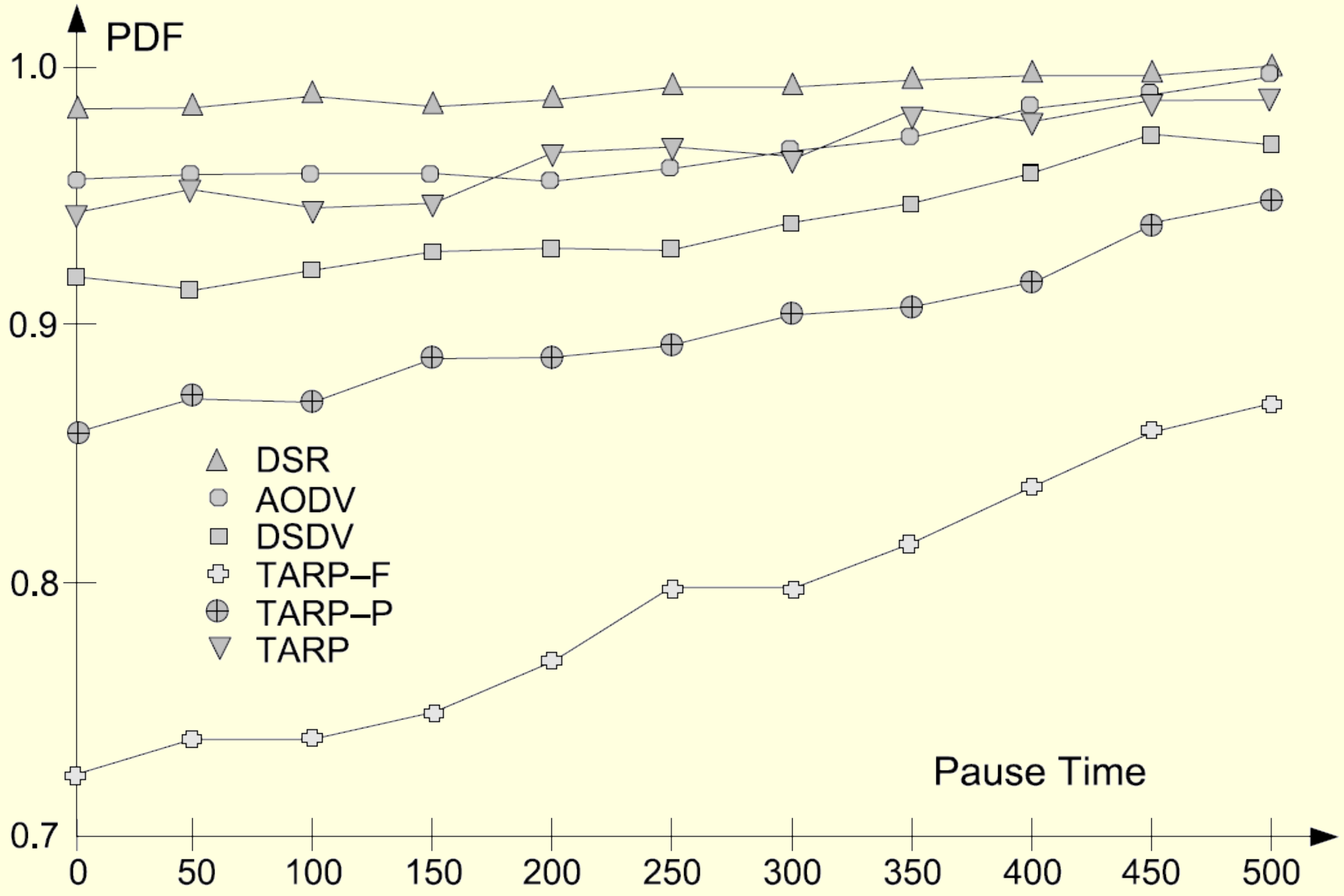
- An instance of cross-layering. The “fuzzy ACK” is sent upon network layer decision to forward the packet. It essentially absolves the sender of the responsibility of handling the packet.

Additional Considerations (cont'd)



- Multiple paths with equal length but within range of each other. Set *opf* flag when the SPD fails (non-forcibly). If *opf* is set, the DD rule compares against packet queued for transmission. If the queued h_f is not less than h_f-1 of the received duplicate, the packet is dropped.

TARP Performance



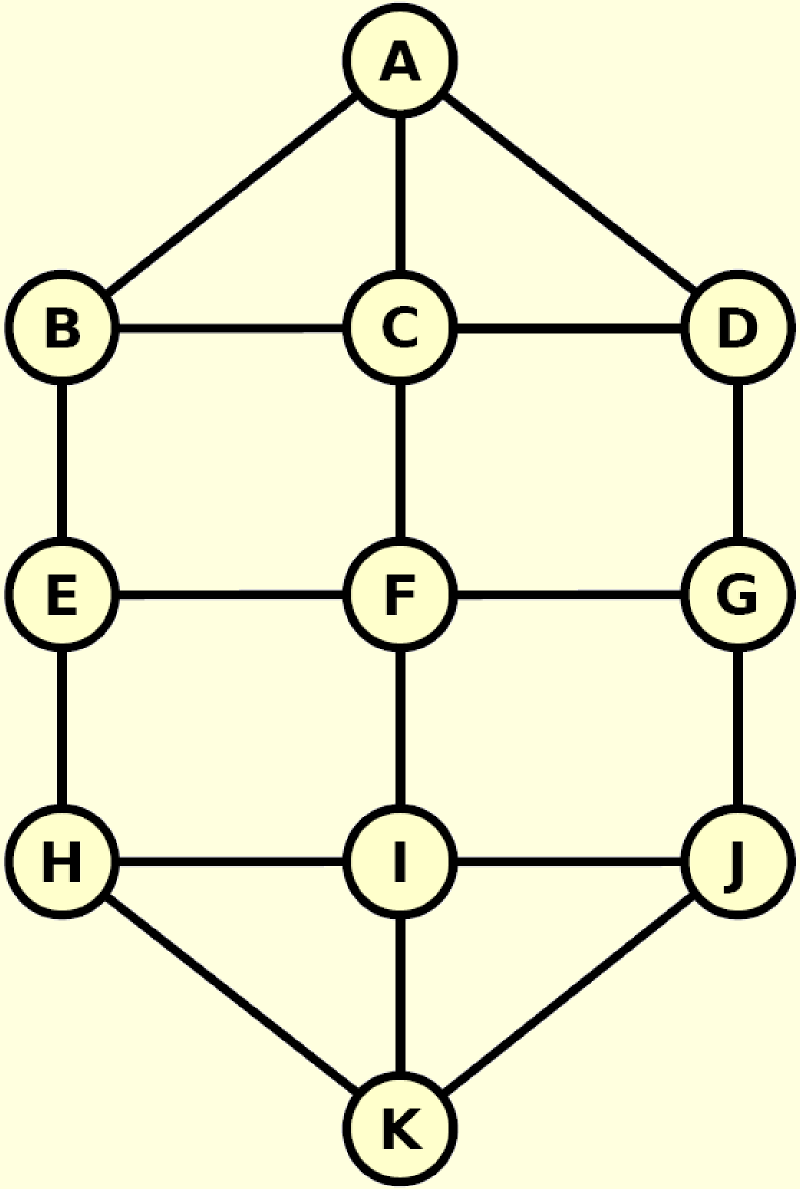
Spanning Structures

- (Mainly) Spanning Trees

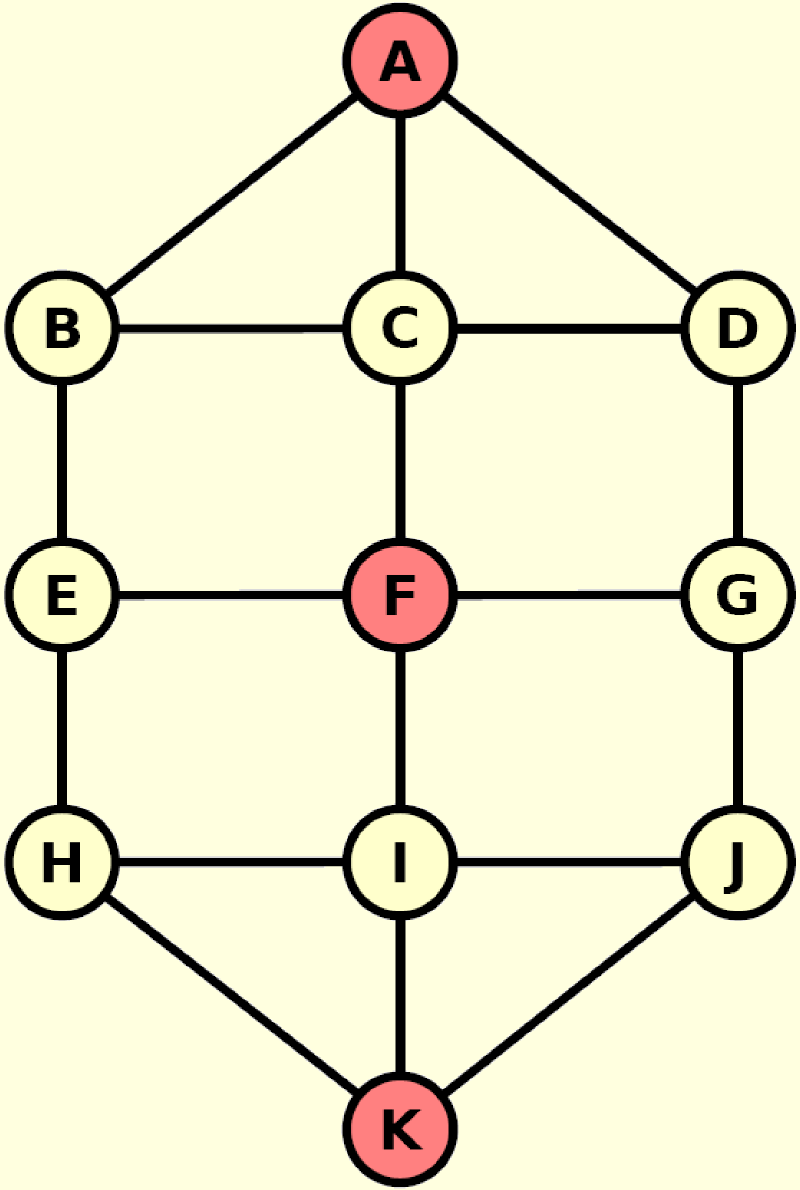
- Features:
  - All nodes reachable.
    - Basic ingredient for data collection and reporting.
  - Ordering via the parent/child relationship.
  - Logical separation of “interior” and “leaf” nodes.
  - Restricted forms possible.
    - For example spanning within a geographic area.
  - Spanning tree can be tuned to application needs.
    - What is the most useful spanning tree?

Clusters and Spanning Structure

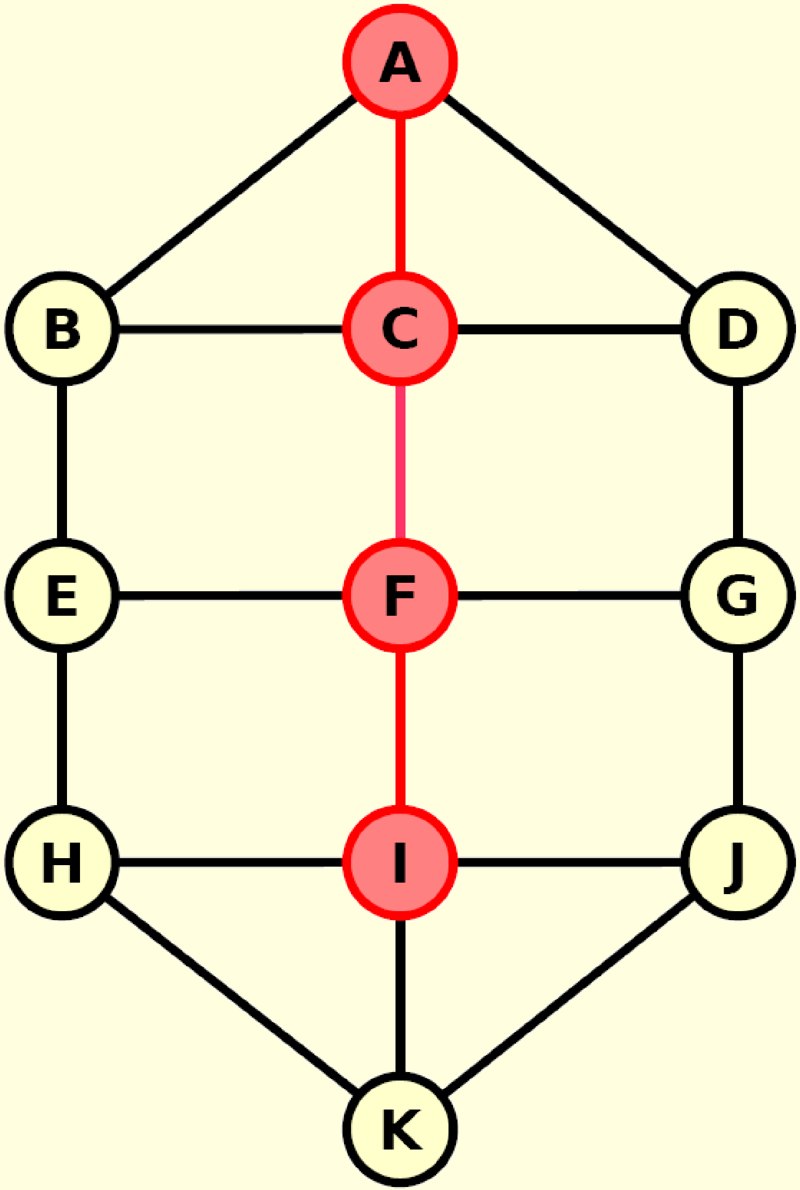
- Name the interior nodes *cluster-heads* and the leaf nodes as cluster *members*.
- The vast majority of cluster-based logical structures present in the literature provide **also** a strategy for **connecting** the clusters.
- Translate the what is a “good spanning structure” question to what is a “good clustering structure”.
- Hence, check out the (Connected) Dominating Set.



An example physical topology.



A Dominating Set.



A Connected Dominating Set.
(4 transmissions)

CDS Spanning Structures

- We are interested in the Minimum CDS (MCDS)
  - It represents the minimum number of transmissions to “get to all nodes.”
  - An NP-hard problem (even on UDGs) with some known approximations including distributed ones.
- MCDS has received attention already:
  - Routing in mobile ad-hoc networks: OLSR
  - Multicasting in mobile ad-hoc networks, etc.
  - As the other side of the coin: “Leafy” Spanning Trees
- Is MCDS a useful “universal” spanning structure?

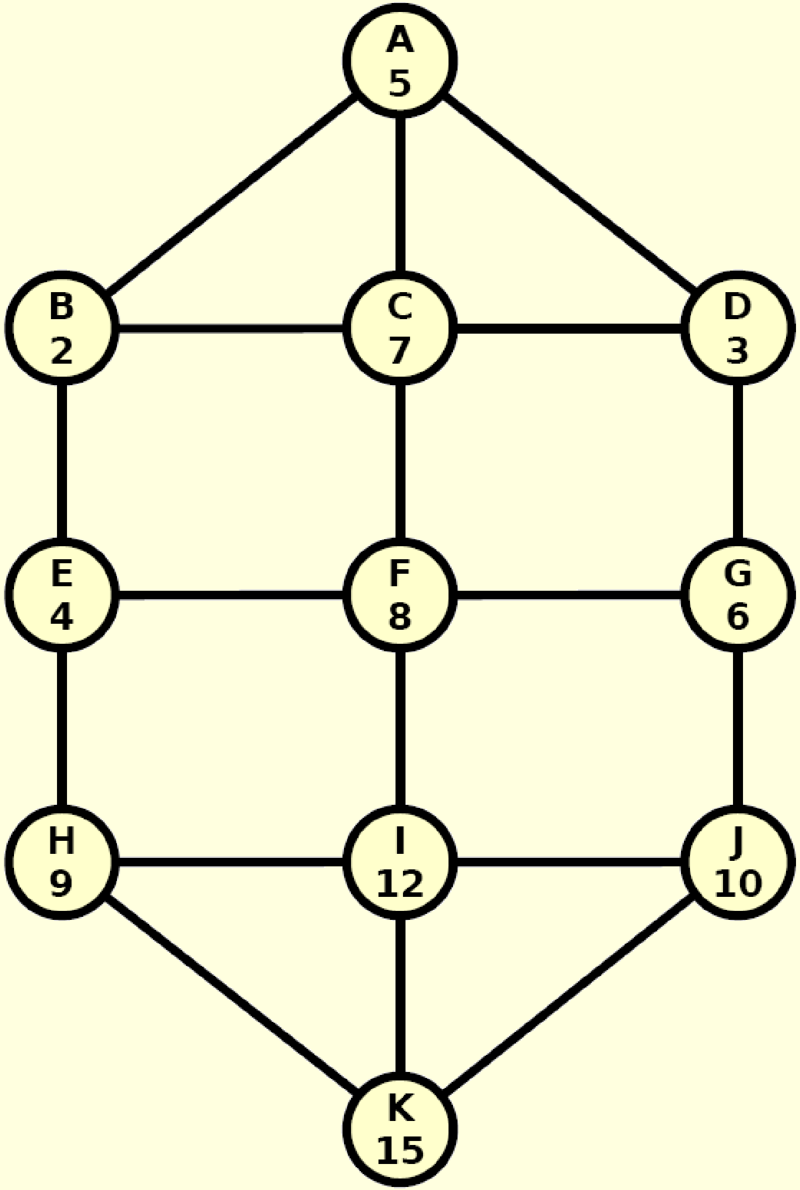
An Example: Top-*k* Query

- Collect (periodically) information from sensors such that the top-*k* values can be determined.
- A Database Sensor Query Processing favorite.
  - Easily expresses min-*k*, max, and min.
- The literature calls for a spanning structure (root=sink) without caring about its characteristics.
  - Usually what is used is a minimum spanning tree, or a shortest path tree (SPT).
  - We will compare against a Dominating Set Tree (DST) constructed as approx.(MCDS)U{Root}

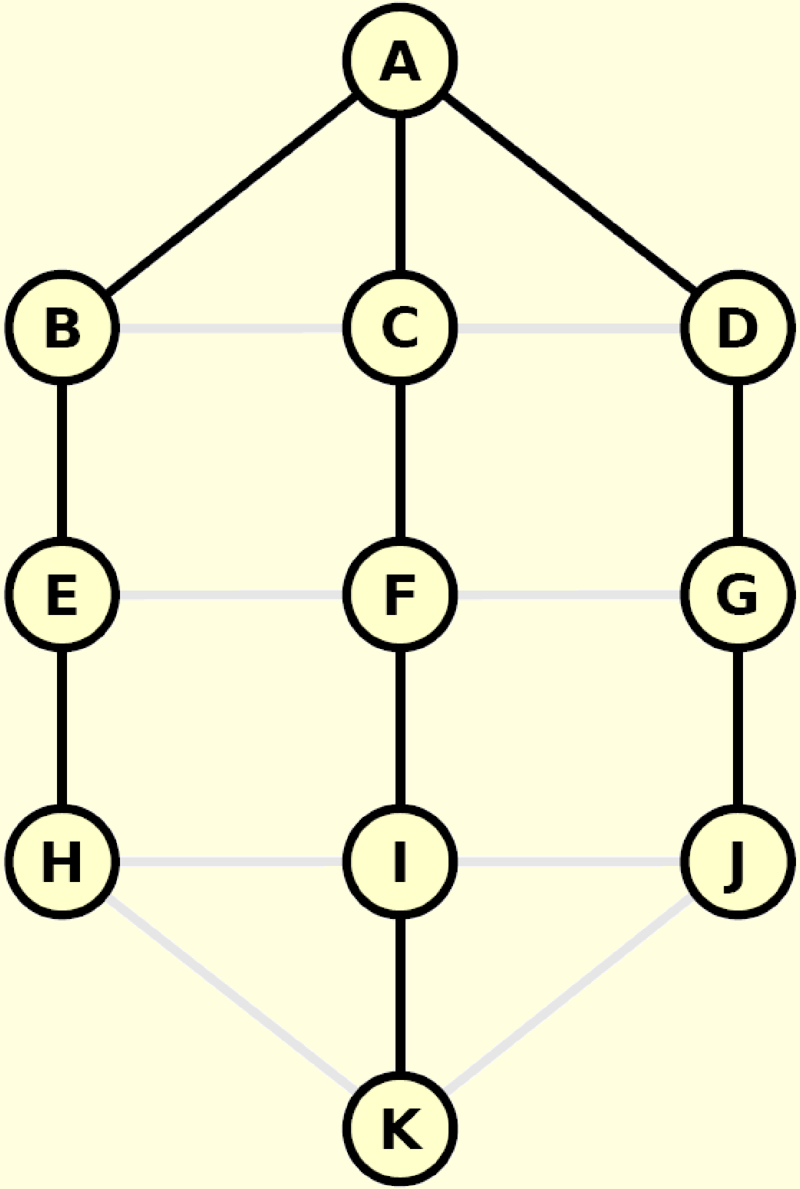
TAG

TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks, by Madden, Franklin, Hellerstein, and Hong (OSDI 2002)

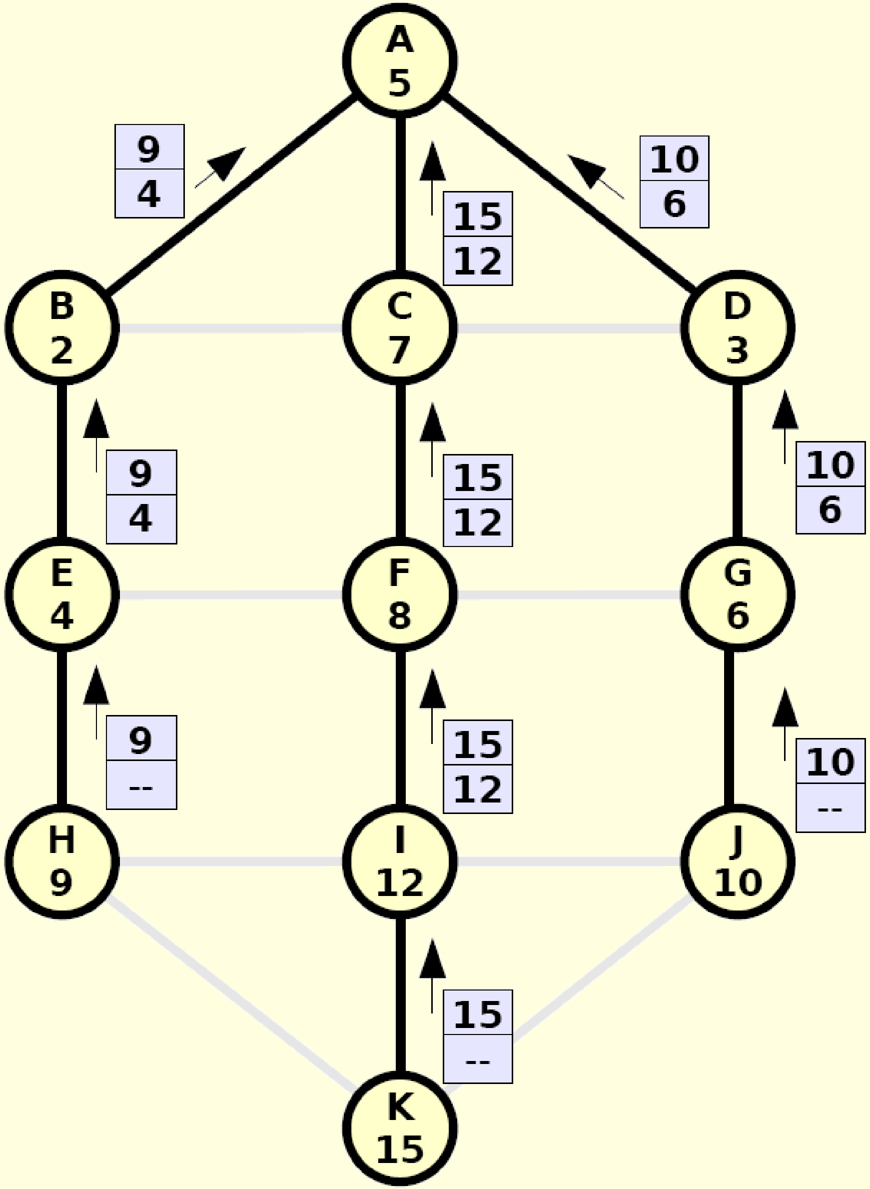
- Intuitive algorithm: perform aggregation on the way from the source to the sink.
- A non-filtering approach. One-shot execution. Repeats in every round, exactly the same way.
- No particular attention to the spanning structure used for the data collection.



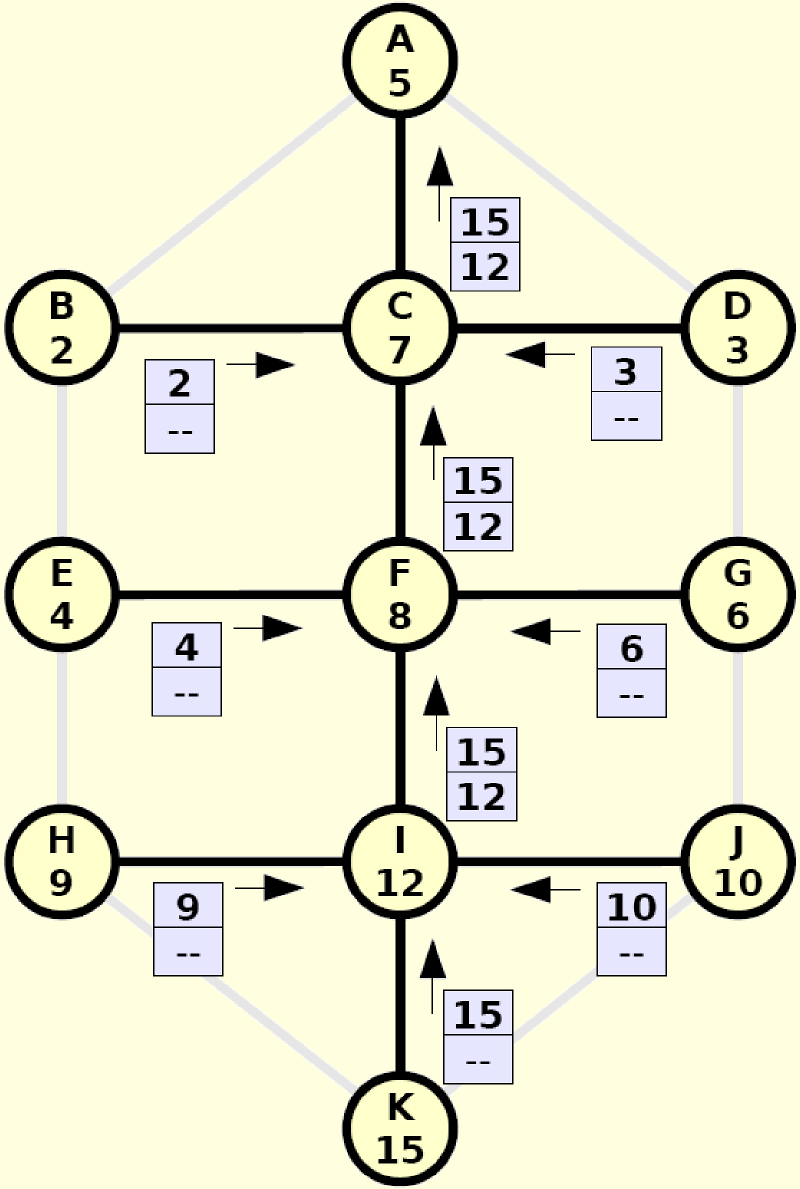
Let A be the root/sink.
Collect the top-2 values at A.



An SPT spanning tree.
(8 messages)



TAG [Madden at al., 2002] version of top-*2* on SPT. (17 message units)

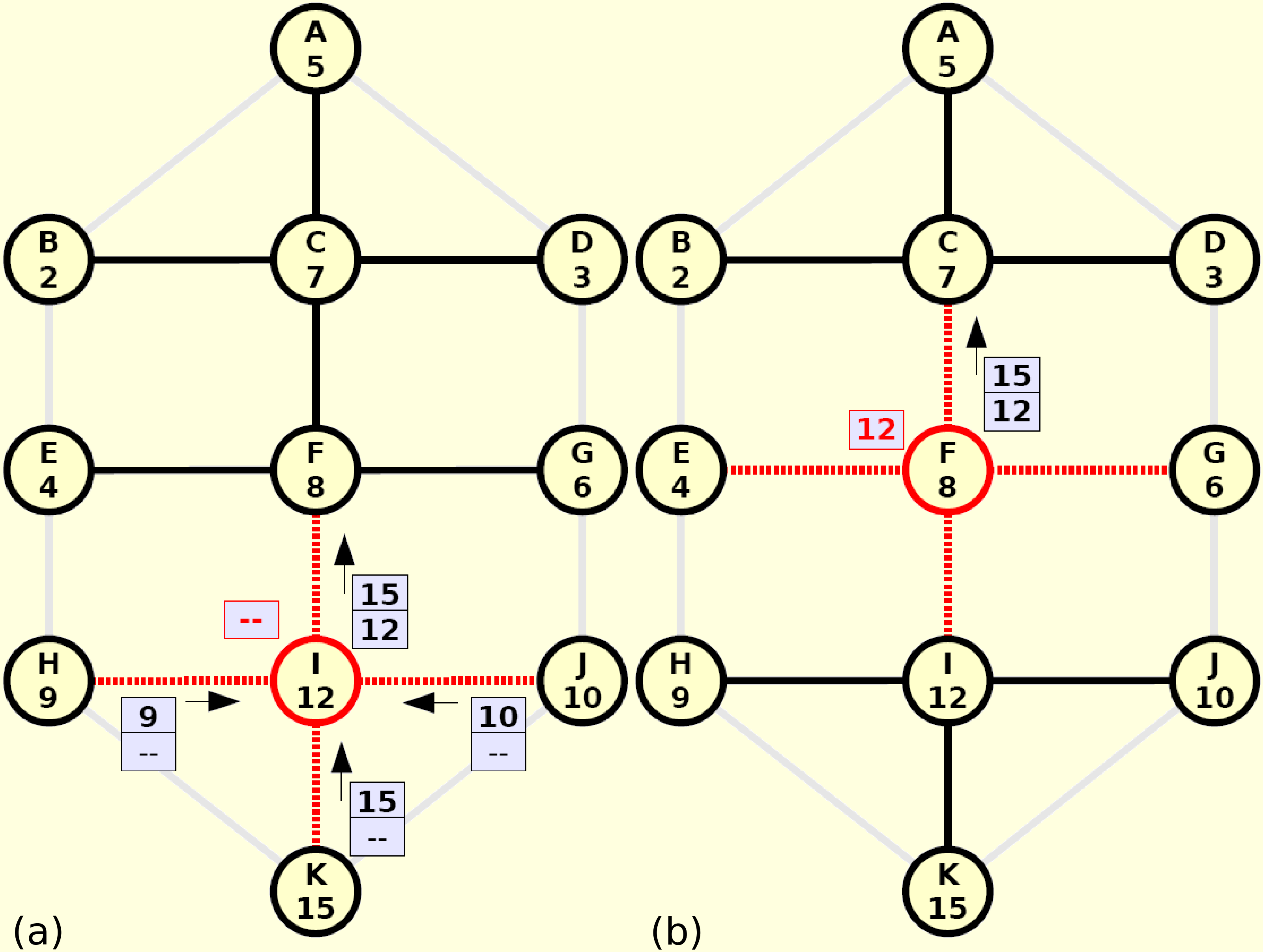


TAG [Madden at al., 2002] version of top-*2* on CDS. (13 message units)

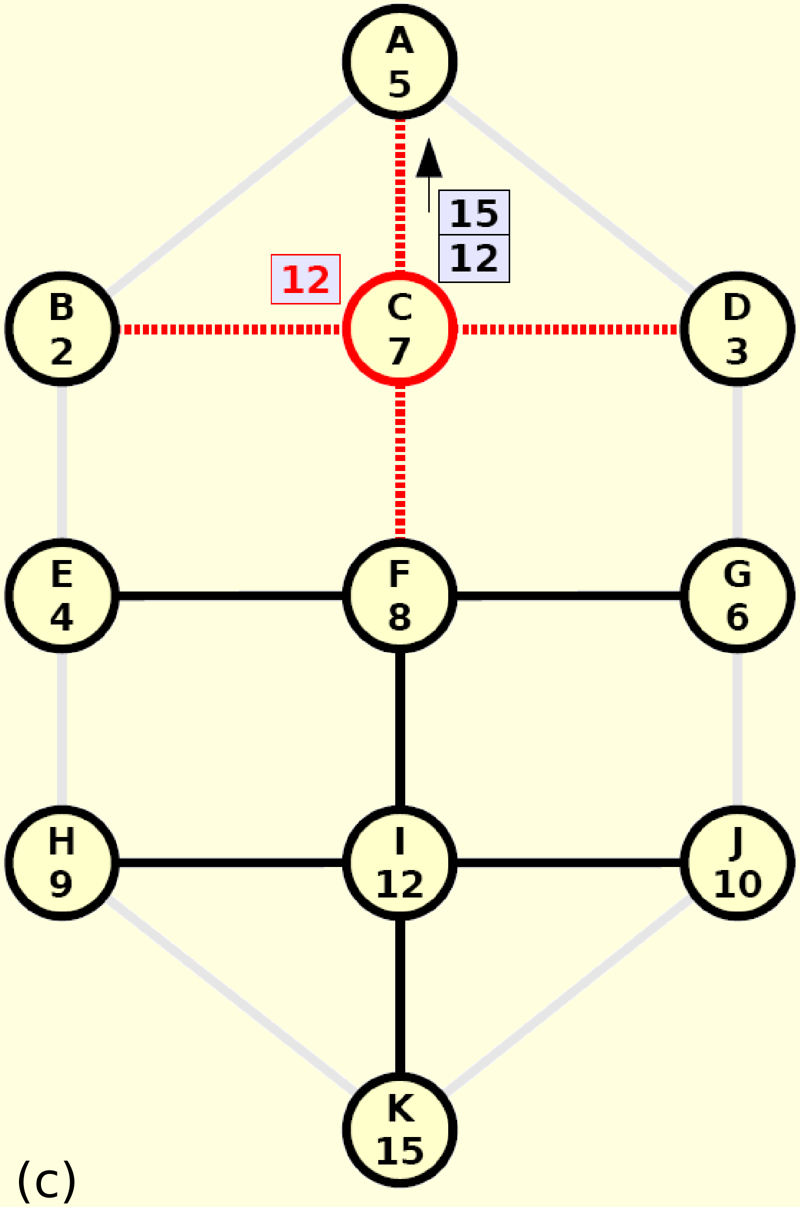
TAGP = TAG + Pruning

- Provide a threshold value when requesting sensor values. The top-*k* is guaranteed to be above the threshold.
- A sensor need not respond (*) if its value is below the provided threshold.
- Refine (raise) the threshold as you traverse toward the sink.

(*) really we need scheduling and/or short responses



A 5
B 2
C 7
D 3
E 4
F 8
G 6
H 9
I 12
J 10
K 15
--
15
12
9
--
10
--
15
--
(a)
A 5
B 2
C 7
D 3
E 4
F 8
G 6
H 9
I 12
J 10
K 15
12
15
12
(b)



TAGP of top-*2* on CDS.
(12 message units)

Filtering-Based Approaches

- Useful for amortizing cost across multiple rounds.
- Intuition: temporal locality, i.e., the set of sensors that gave the top-*k* values will likely be the same giving the top-*k* values in the next round.
- Strategy: opt for a costly re-evaluation only if there has been a (significant?) change in values.
- Semantics: we insist that a top-*k* query means the *k* highest **values** with the possibility that many nodes might be in a tie for each of these *k* values.

Previous Filtering Approaches

Top-k Monitoring in Wireless Sensor Networks, by Wu, Xu, Tang, and Lee (IEEE Trans. KDE, 2007).

- Introduced the FILA protocol.
- Somewhat odd semantics:
  - Tracks the sensors with the *k* highest values, but *not* their values (only approximately). No ties supported.

Energy-Efficient Monitoring of Extreme Values in Sensor Networks, by Silberstein, Braynard, and Yang (SIGMOD, 2006).

- Introduced ATA (Adaptive Threshold Algorithm).
- Defined for top-*1*, i.e., max (min) query only.

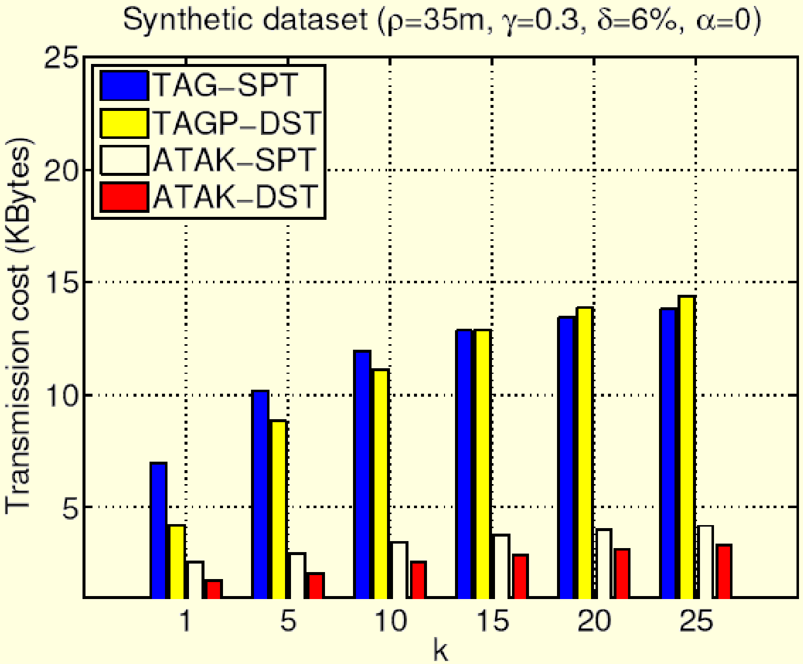
ATA for Top-*k* (ATAK)

- The sensors that are part of the top-*k* in the current round **must** report in the next round if their value **changes**.
- The sensors with values below the top-*k* values are given a threshold (*min* of top-*k* values). They **must** report if they **cross** the threshold.
- Broadcasts (along the spanning structure) at the end of each round, inform all nodes of the new threshold.

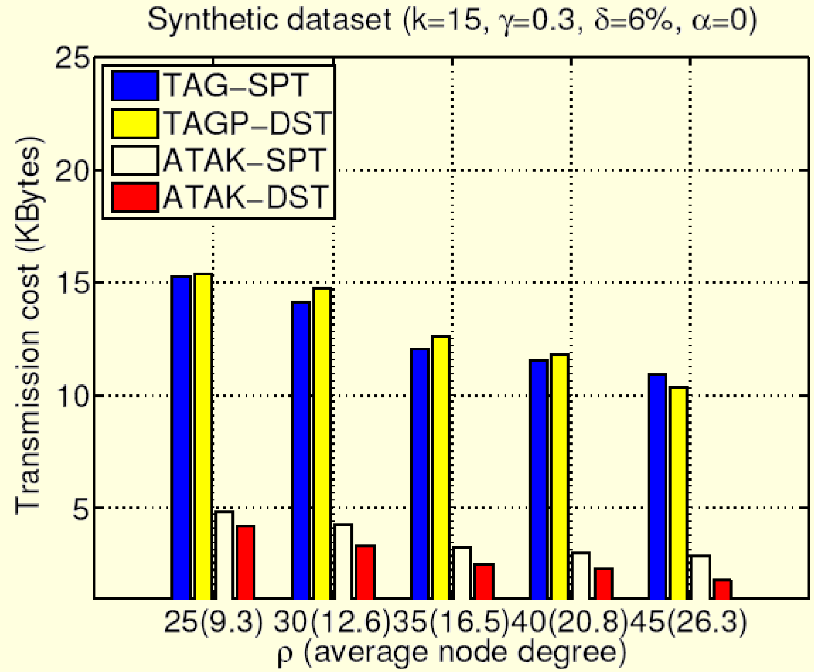
ATAK Technical Details

- We have enough information to recompute the top-*k* values (and corresponding sensors with each one of the top-*k* values) with some exceptions.
- Summary of exceptions:
  - (New) ties in the top-*k* space result in *k*' (<*k*) values "surviving" in the next round, forcing a one-shot top-*(k-k')* query to be executed (over the non-top-*k* values, facilitated by the spanning structure).
  - If more nodes "vacated" the top-*k* space (say *m*) than moved from non-top-*k* to top-*k,* we need find all values (from non-top-*k* space) that are greater than the *(k-k')*-th highest value from the set of *m* values that "dipped" below the threshold (facilitated by the spanning structure).

Message Overhead



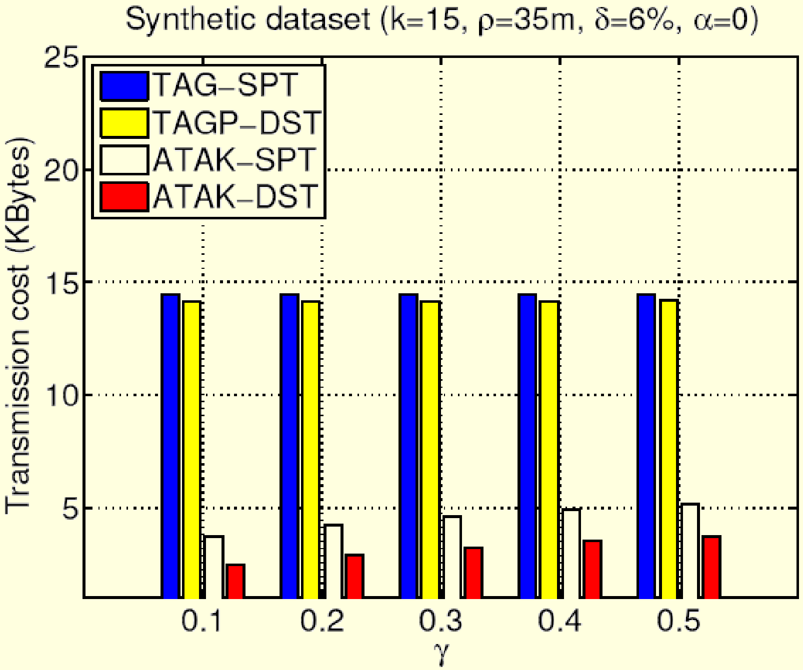
(a) Varying k



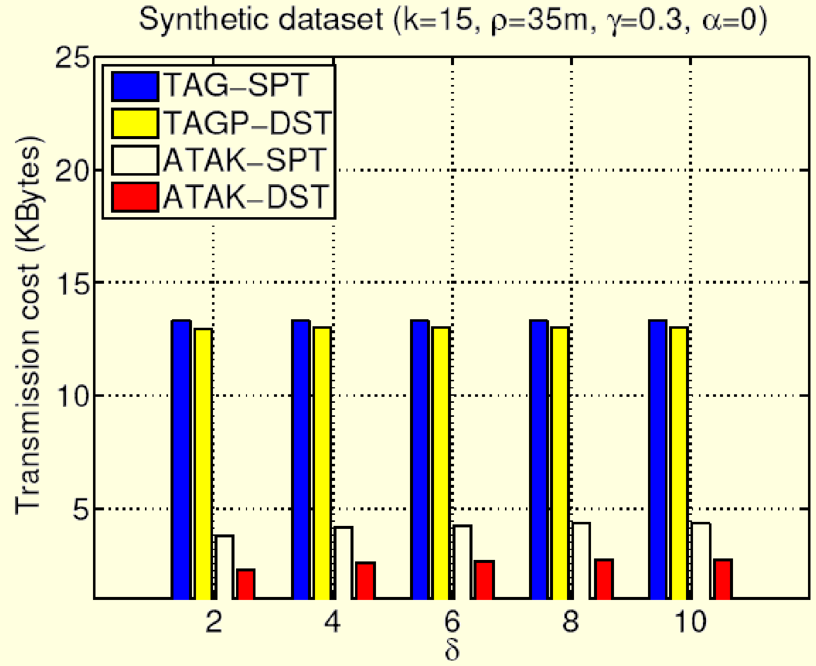
(b) Varying ρ

ρ = radio range

Message Overhead (cont'd)



(c) Varying γ

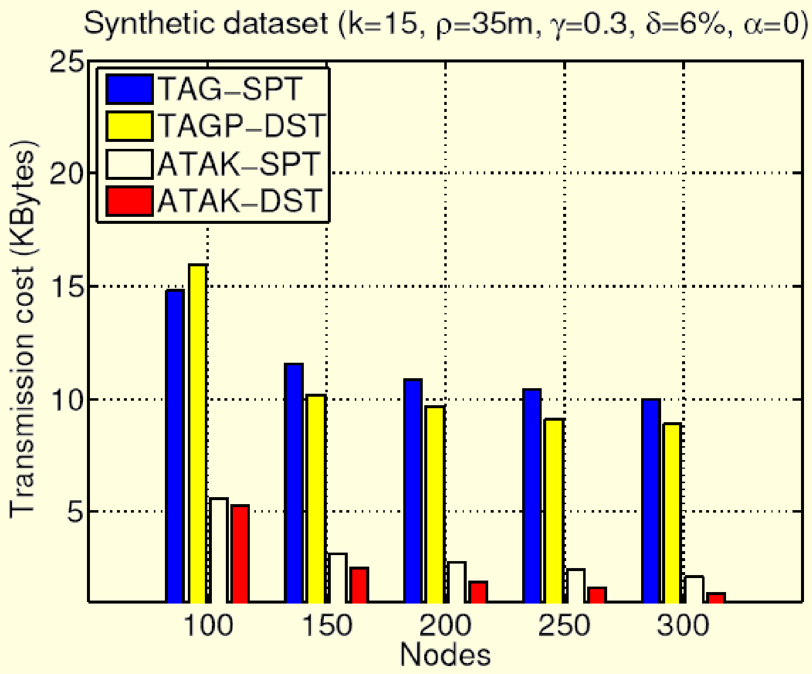


(d) Varying δ

γ = prob of change

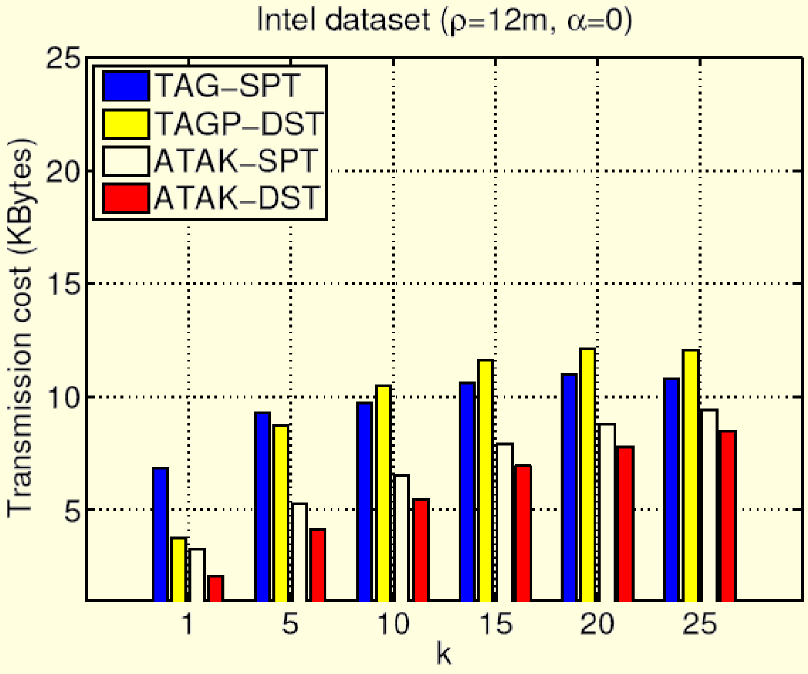
δ =% of change

Message Overhead (cont'd)

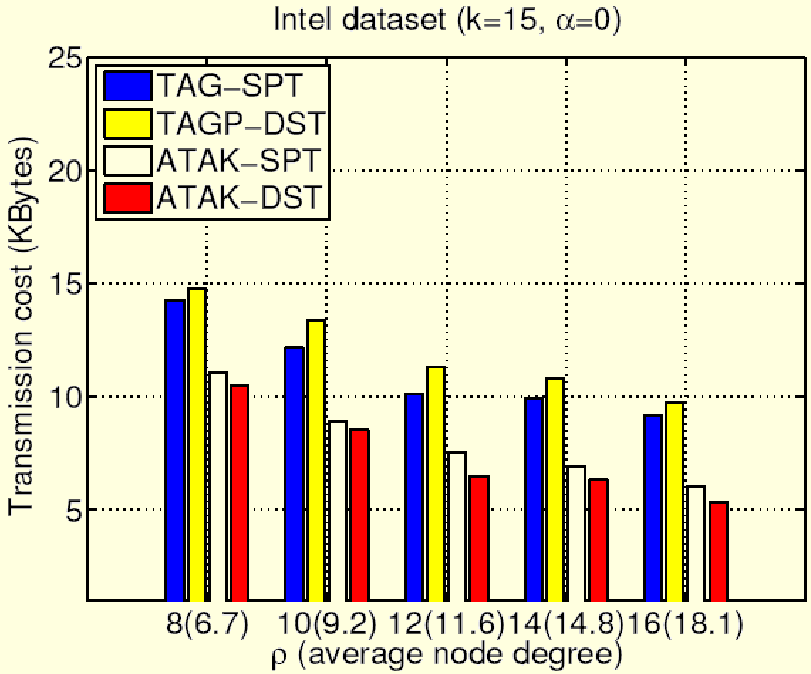


(e) Varying number of nodes

Message Overhead (cont'd)



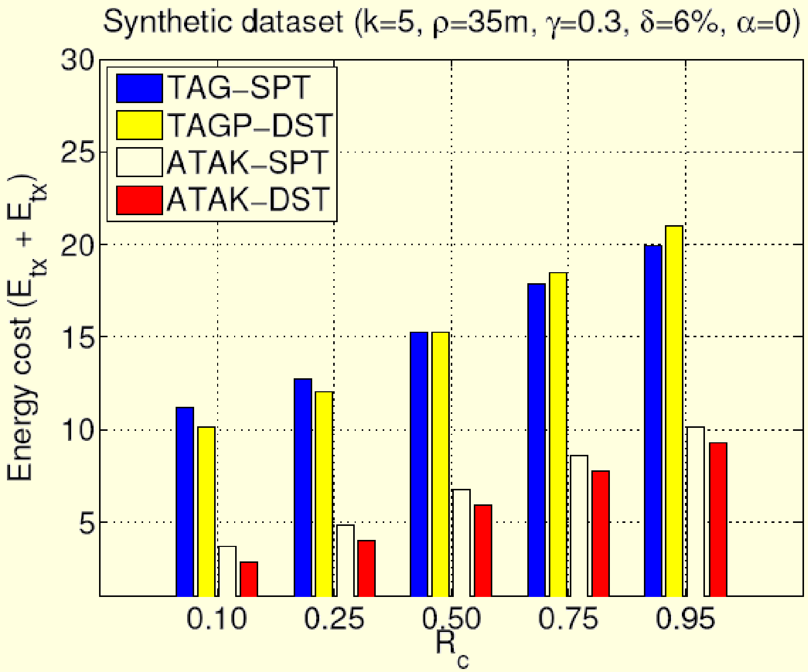
(a) Varying k



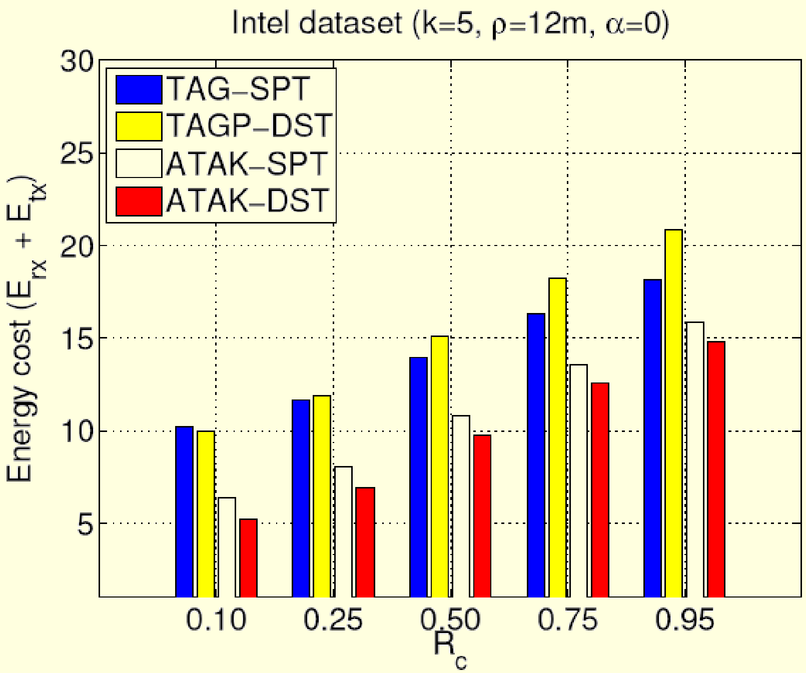
(b) Varying ρ

Intel dataset.

Energy Cost



(a) Synthetic dataset



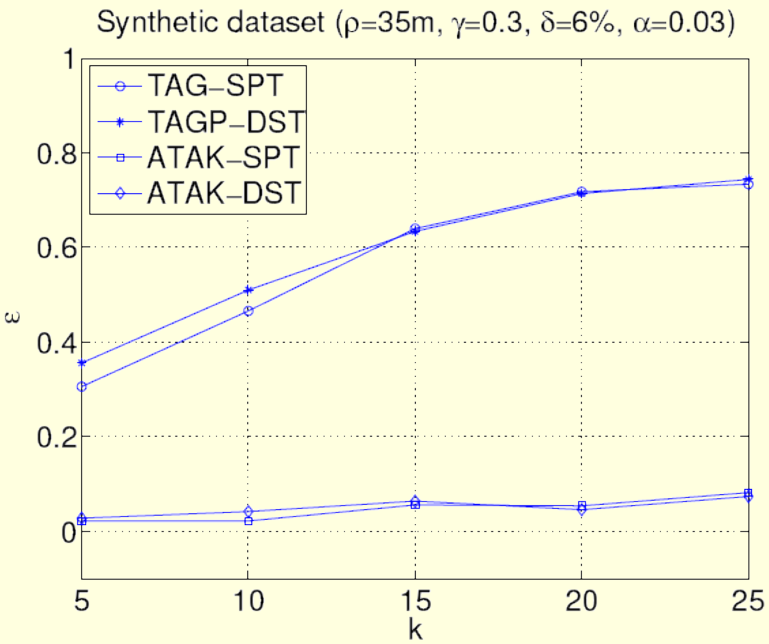
(b) Intel dataset

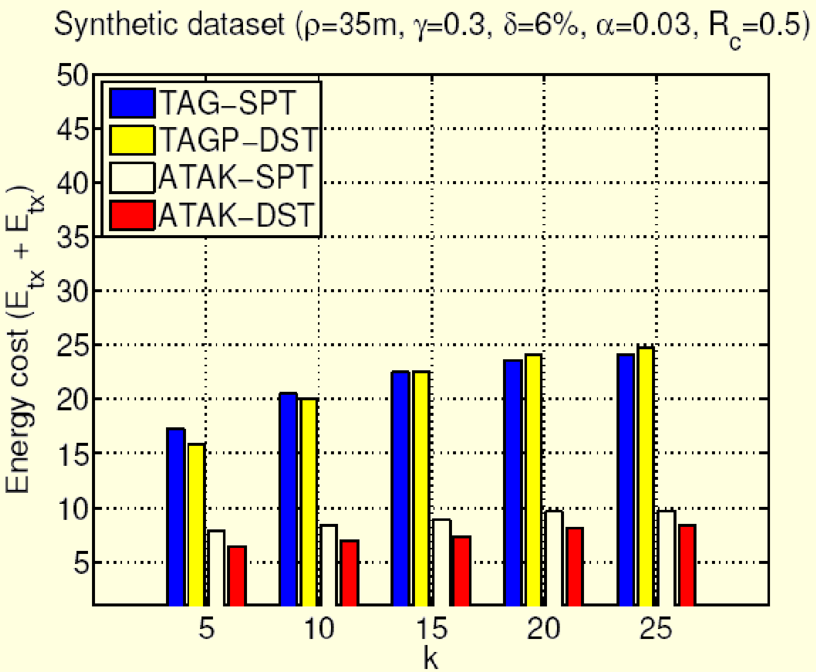
$$R_c = E_r/E_t$$

Dealing with (Rare) Failures

- Node failures: no real options
  - If non-leaf, then reconstruct tree.
- Link failures: exploit top-*k* semantics
  - Opportunistically, attempt (locally) to determine if the failure has distorted the result. If yes, using a unicast (“shortcut”) alert the sink and send the value(s) that were not seen.
- Opportunistic failure determination
  - If we overheard the parent transmit an inconsistent result, then alert sink.
  - If the parent's transmission was not received, then ask the neighbors if they heard the parent's transmission. (Have at least 2-hop neighbor info.)
  - What gets in the way: scheduling (not solved).

Application Error Rate and Energy Cost





α = (individual) link error rate

ε = Application error rate

(Fraction of rounds with incorrect top-*k* results.)

Lessons

- A spanning structure based on an underlying “clustering” (like MCDS) introduces sensitivity to the density and scale of the network that might not be a good match for “naive” algorithms if the cluster does not perform “aggressive” reduction of data volume.
- A good use of an MCDS-based tree is when it is known that the protocol has to employ broadcasts for “state update” (like the threshold).
- But broadcasts should be used sparingly, so a suggested match would be for algorithms where broadcasts are “amortizable” over longer time frames.

Instead of Conclusions: Pie-in-the-sky

- Provide (web) applications access to sensor data by transforming sensors to “first class citizens” of the cyber-infrastructure (à la SOA).
- Technical Issues: reliability, security, routing, energy consumption, hostile deployment environments, long-term maintenance.
- Challenges:
  - abstractions (expressions of elementary protocols) & interface layer to express elementary sensor behaviors to make them accessible to programmers, ...
  - ... and we did not even touch the need for multi-tiered code distribution tools & architecture

Thanks

NSERC
CRSNG

Alberta Advanced Education and Technology

olso net