# Ad hoc and Sensor Networks
# Chapter 12: Data-centric and content-based networking

# Goal of this chapter

- Apart from routing protocols that use a direct identifier of nodes (either unique id or position of a node), networking can talk place based directly on *content*
- Content can be collected from network, processed in the network, and stored in the network
- This chapter looks at such *content-based networking* and *data aggregation* mechanisms

# Overview

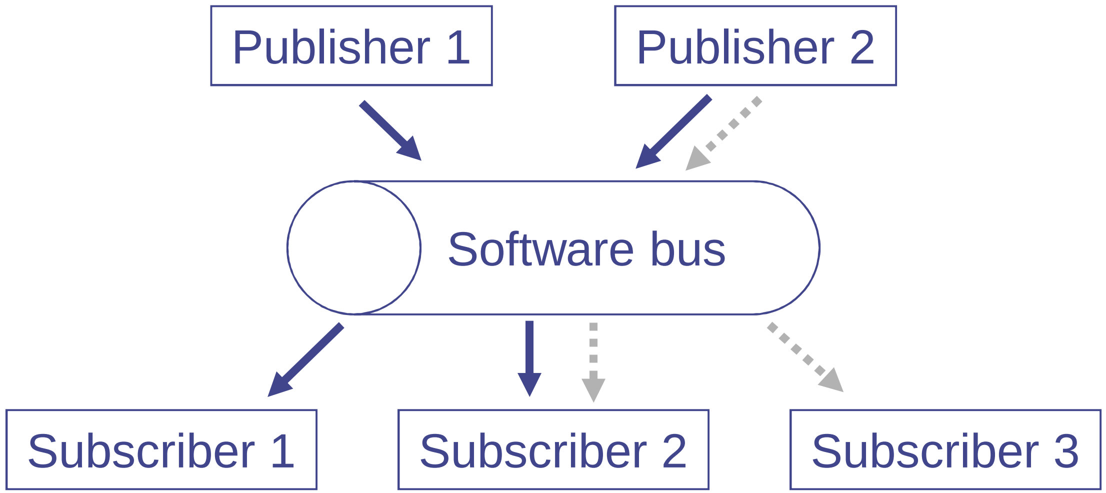
- ***Interaction patterns and programming model***
- Data-centric routing
- Data aggregation
- Data storage

# Desirable interaction paradigm properties

- Standard networking interaction paradigms: Client/server, peer-to-peer
  - Explicit or implicit partners, explicit cause for communication
- Desirable properties for WSN (and other applications)
  - ***Decoupling in space*** – neither sender nor receiver need to know their partner
  - ***Decoupling in time*** – “answer” not necessarily directly triggered by “question”, asynchronous communication

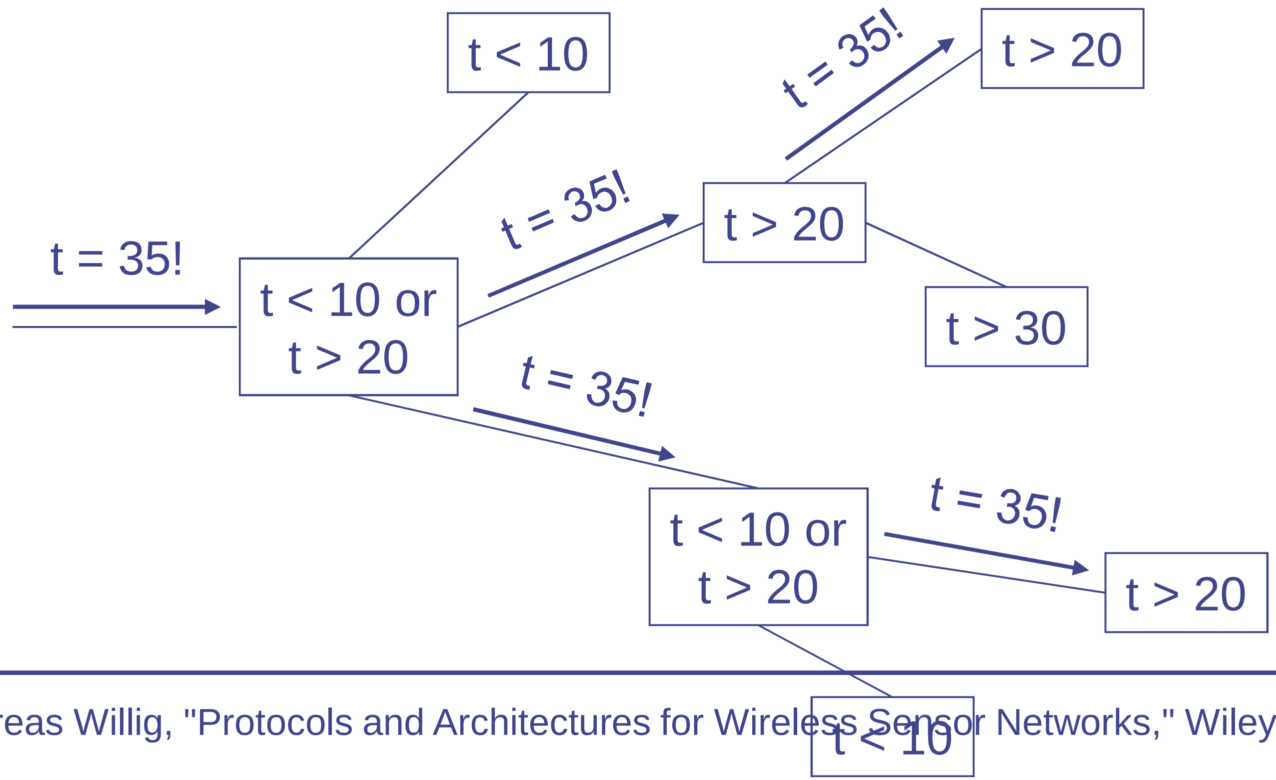
# Interaction paradigm: Publish/subscribe

- Achieved by ***publish/subscribe*** paradigm
  - Idea: Entities can publish data under certain names
  - Entities can subscribe to updates of such ***named data***
- Conceptually: Implemented by a software bus
  - Software bus stores subscriptions, published data; names used as filters; subscribers notified when values of named data changes
- Variations
  - ***Topic-based*** P/S – inflexible
  - ***Content-based*** P/S – use general predicates over named data



# Publish/subscribe implementation options

- Central server – mostly not applicable
- Topic-based P/S: group communication protocols
- Content-based networking does not directly map to multicast groups
  - Needs content-based routing/forwarding for efficient networking



# Overview

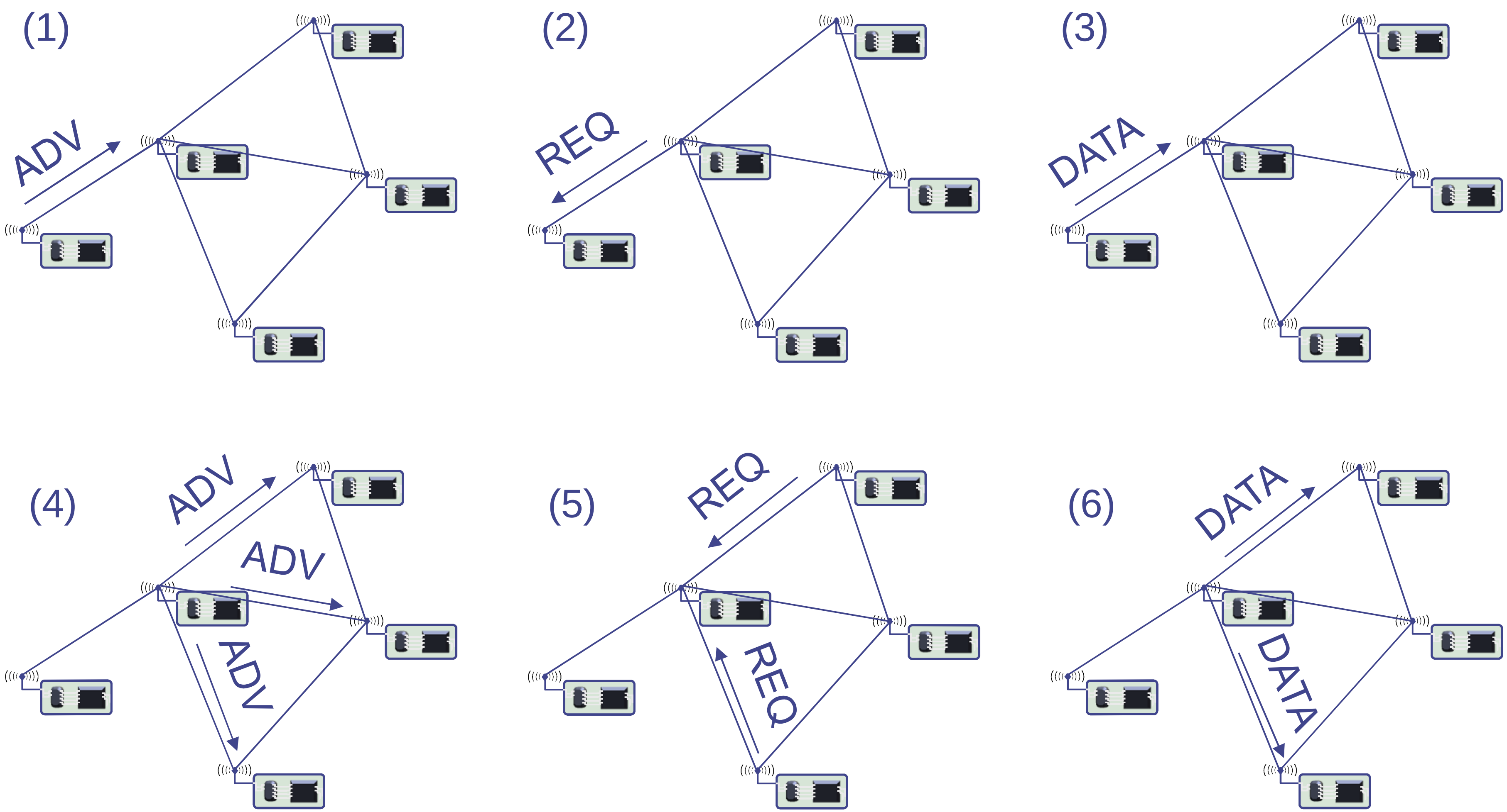
- Interaction patterns and programming model
- ***Data-centric routing***
- Data aggregation
- Data storage

# One-shot interactions with big data sets

- Scenario
  - Large amount of data are to be communicated – e.g., video picture
  - Can be succinctly summarized/described
- Idea: Only exchange characterization with neighbor, ask whether it is interested in data
  - Only transmit data when explicitly requested
  - Nodes should know about interests of further away nodes

! ***Sensor Protocol for Information via Negotiation*** (SPIN)

# SPIN example



# Repeated interactions

- More interesting: Subscribe once, events happen multiple times
  - Exploring the network topology might actually pay off
  - But: unknown which node can provide data, multiple nodes might ask for data

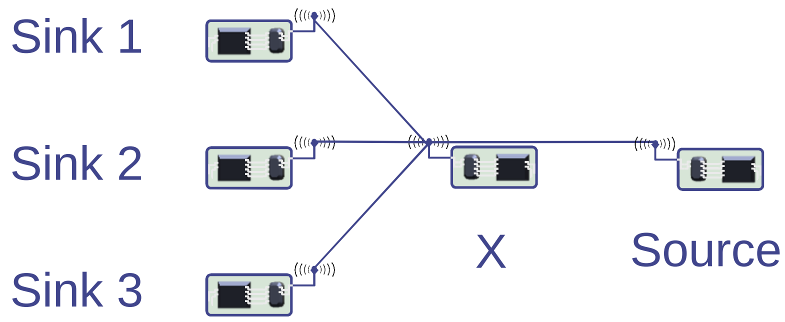
  ! How to map this onto a “routing” problem?
- Idea: Put enough information into the network so that publications and subscriptions can be mapped onto each other
  - But try to avoid using unique identifiers: might not be available, might require too big a state size in intermediate nodes

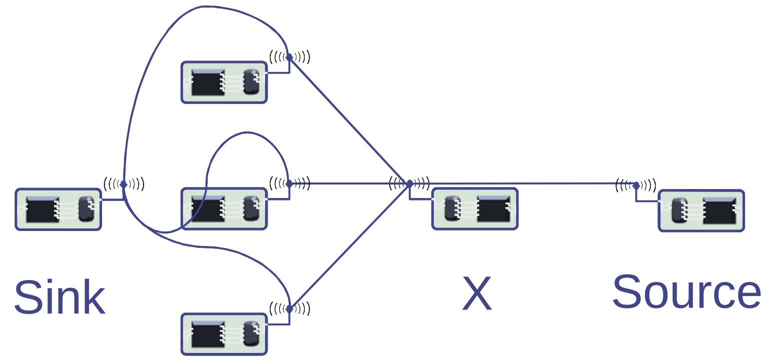
! ***Directed diffusion*** as one option for implementation

  - Try to rely only on ***local interactions*** for implementation

# Directed diffusion – Two-phase pull

- **Phase 1**: nodes distribute ***interests*** in certain kinds of named data
  - Specified as attribute-value pairs (cp. Chapter 7)
- Interests are flooded in the network
  - Apparently obvious solution: remember from where interests came, set up a convergecast tree
  - Problem: Node X cannot distinguish, in absence of unique identifiers, between the two situations on the right – set up only one or three convergecast trees?



# Direction diffusion – Gradients in two-phase pull

- Option 1: Node X forwarding received data to all "parents" in a "convergecast tree"
  - Not attractive, many needless packet repetitions over multiple routes
- Option 2: node X only forwards to one parent
  - Not acceptable, data sinks might miss events
- Option 3: Only provisionally send data to all parents, but ask data sinks to help in selecting which paths are redundant, which are needed
  - Information from where an interest came is called ***gradient***
  - Forward all published data along all existing gradients

# Gradient reinforcement

- Gradients express not only a link in a tree, but a quantified "strength" of relationship
  - Initialized to low values
  - Strength represents also rate with which data is to be sent
- Intermediate nodes forward on all gradients
  - Can use a data cache to suppress needless duplicates
- **Second phase**: Nodes that contribute new data (not found in cache) should be encouraged to send more data
  - Sending rate is increased, the gradient is ***reinforced***
  - Gradient reinforcement can start from the sink
  - If requested rate is higher than available rate, gradient reinforcement propagates towards original data sources
- Adapts to changes in data sources, topology, sinks

# Directed diffusion – extensions

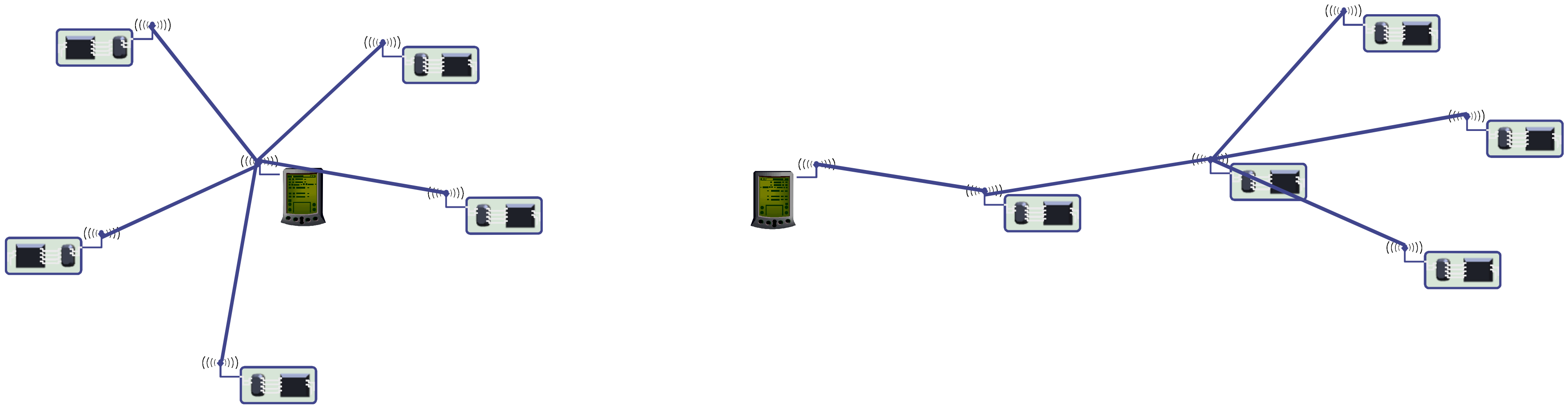
- Two-phase pull suffers from interest flooding problems
  - Can be ameliorated by combining with topology control, in particular, passive clustering
- Geographic scoping & directed diffusion
- Push diffusion – few senders, many receivers
  - Same interface/naming concept, but different routing protocol
  - Here: do not flood interests, but flood the (relatively few) data
  - Interested nodes will start reinforcing the gradients
- Pull diffusion – many senders, few receivers
  - Still flood interest messages, but directly set up a real tree

# Overview

- Interaction patterns and programming model
- Data-centric routing
- ***Data aggregation***
- Data storage

# Data aggregation

- Any packet not transmitted does not need energy
- To still transmit data, packets need to combine their data into fewer packets ! ***aggregation*** is needed
- Depending on network, aggregation can be useful or pointless



# Metrics for data aggregation

- ***Accuracy***: Difference between value(s) the sink obtains from aggregated packets and from the actual value (obtained in case no aggregation/no faults occur)
- ***Completeness***: Percentage of all readings included in computing the final aggregate at the sink
- ***Latency***
- ***Message overhead***

# How to express aggregation request?

- One option: Use database abstraction of WSN
- Aggregation is requested by appropriate SQL clauses

```
SELECT {agg(expr), attributes} FROM sensors
WHERE {selectionPredicates}
GROUP BY {attributes}
HAVING {havingPredicates}
EPOCH DURATION i
```

- Agg(expr): actual aggregation function, e.g., AVG(temperature)
- WHERE: filter on value before entering aggregation process
  - Usually evaluated locally on an observing node
- GROUP BY: partition into subsets, filtered by HAVING
  - GROUP BY floor HAVING floor > 5

# Partial state records

- Partial state records to represent intermediate results
  - E.g., to compute average, sum and number of previously aggregated values is required – expressed as <sum,count>
  - Update rule: $< s, c > = < s_1 + s_2, c_1 + c_2 >$
  - Final result is simply s/c

# Aggregation operations – categories

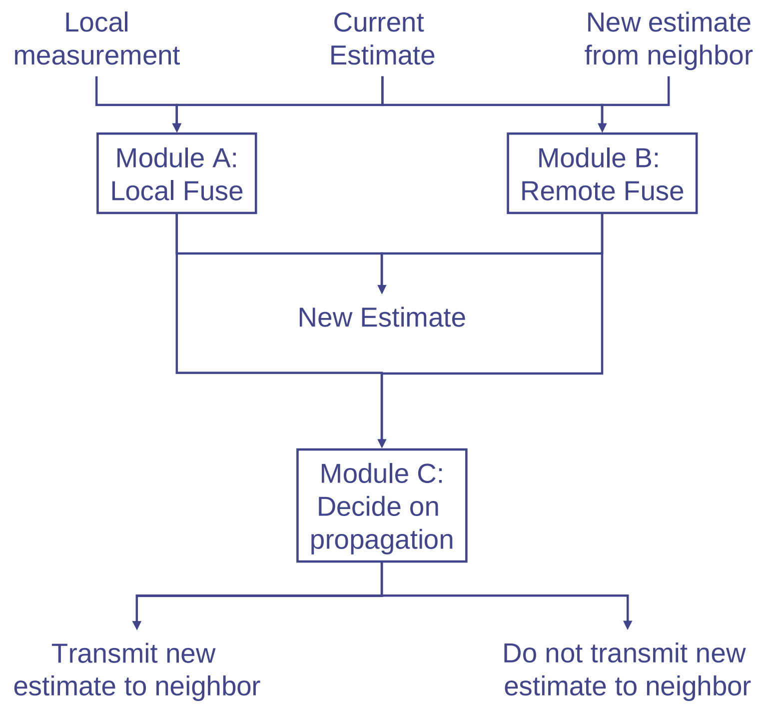
- Duplicate sensitive, e.g., median, sum, histograms; insensitive: maximum or minimum
- Summary or examplary
- Composable: for f aggregation function, there exist g such that $f(W) = g(f(W_1), f(W_2))$ for $W = W_1 \cap W_2$
- Behavior of partial state records
  - Distributive – end results directly as partial state record, e.g., MIN
  - Algebraic – p.s.r. has constant size; end result easily derived
  - Content-sensitive – size and structure depend on measured values (e.g., histogram)
  - Holistic – all data need to be included, e.g., median
- Monotonic

# Placement of aggregation points

- Convergecast trees provide natural aggregation points
- But: what are *good* aggregation points?
  - Ideally: choose tree structure such that the size of the aggregated data to be communicated is minimized
  - Figuratively: long trunks, bushy at the leaves
  - In fact: again a Steiner tree problem in disguise
- Good aggregation tree structure can be obtained by slightly modifying Takahashi-Matsuyama heuristic
- Alternative: look at parent selection rule in a simple flooding-based tree construction
  - E.g., first inviter as parent, random inviter, nearest inviter, …
  - Result: no simple rule guarantees an optimal aggregation structure
- Can be regarded as optimization problem as well

# Alternative: broadcasting an aggregated value

- Goal is to distribute an aggregate of all nodes' measurements to all nodes in turn
  - Setting up |V| convergecast trees not appropriate
- Idea: Use gossiping combined with aggregation
  - When new information is obtained, locally or from neighbor, compute new estimate by aggregation
  - Decide whether to gossip this new estimate, detect whether a change is "significant"



# Overview

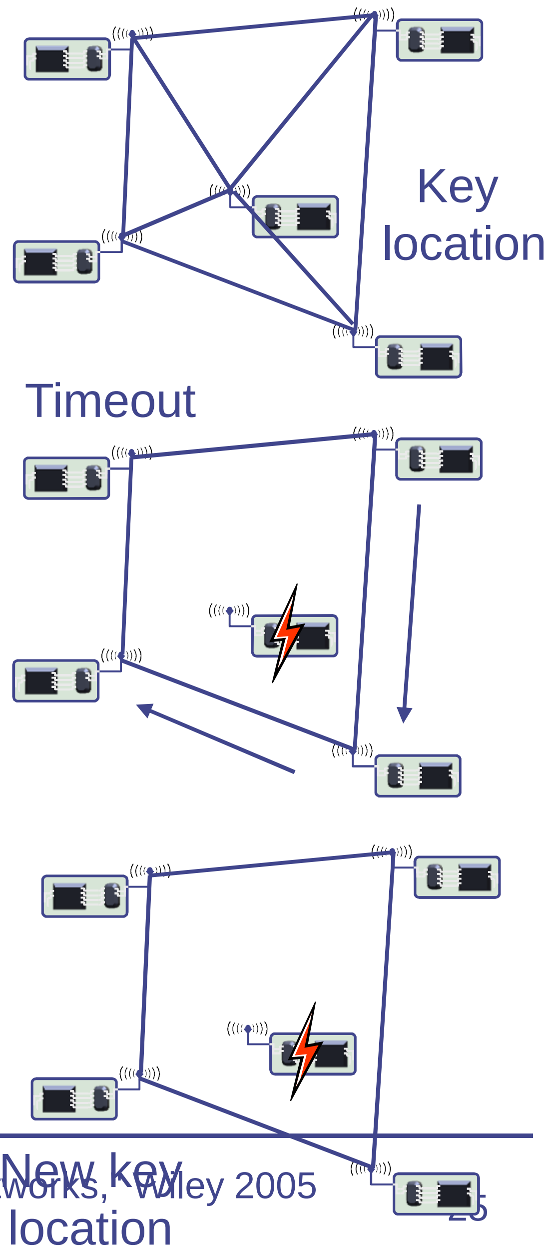
- Interaction patterns and programming model
- Data-centric routing
- Data aggregation
- ***Data storage***

# Data-centric storage

- Problem: Sometimes, data has to be stored for later retrieval – difficult in absence of gateway nodes/servers
- Question: Where/on which node to put a certain datum?
  - Avoid a complex directory service
- Idea: Let name of data describe which node is in charge
  - Data name is hashed to a geographic position
  - Node closest to this position is in charge of holding data
  - Akin to peer-to-peer networking/distributed hash tables
  - Hence name of one approach: ***Geographic Hash Tables*** (***GHT***)
  - Use geographic routing to store/retrieve data at this "location" (in fact, the node)

# Geographic hash tables – Some details

- Good hash function design
- Nodes not available at the hashed location – use "nearest" node as determined by a geographic routing protocol
  - E.g., the node where an initial packet started circulating the "hole"
  - Other nodes around hole are informed about node taking charge
- Handling failing and new nodes
  - Failure detected by timeout, apply similar procedure as for initially storing data
- Limited storage per node
  - Distribute data to other nodes on same face



# Conclusion

- Using data names or predicates over data to describe the destination of packets/data opens new options for networking
- Networking based on such “data-centric addresses” nicely supports an intuitive programming model – publish/subscribe
- Aggregation a key enabler for efficient networking
- Other options – data storage, bradcasting aggregates – also well supportable