AIT

Johannes Lipp

10. April 2014

Inhaltsverzeichnis

Kapitel 1

Peer-to-Peer Systems

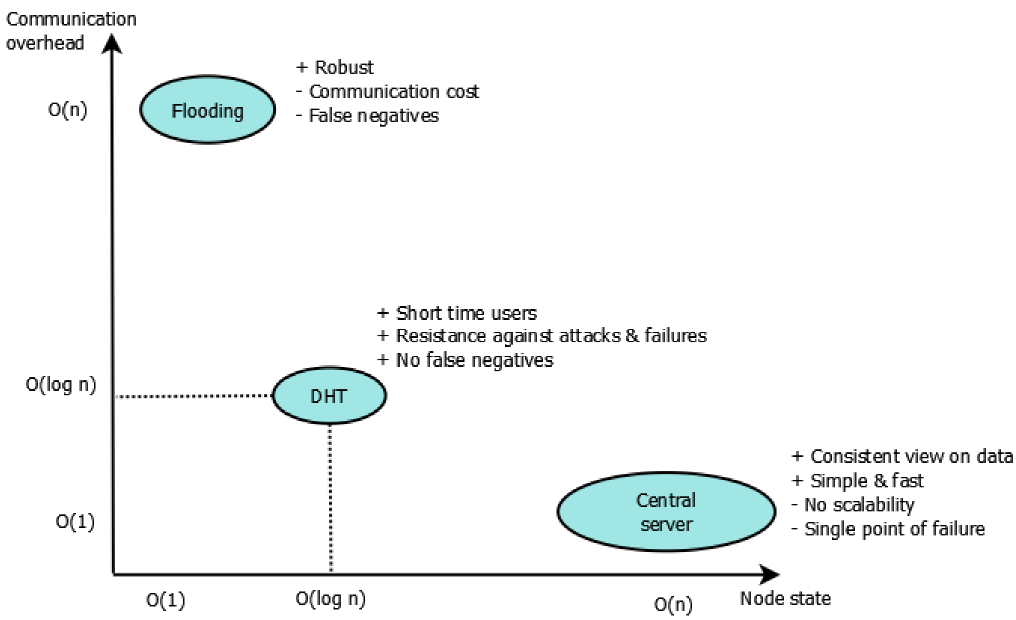
1.1 Unstructured P2P Approaches

1.1.1 Central Server: Napster

1.1.2 Unstructured P2P (flooding)

- Pure P2P: Gnutella 0.4, Freenet
  + No single point of failure
  + Anonymity
  + Fuzzy queries
  - High traffic
  - Overlay topology not optimal
  - Zig-zag routes
  - False negatives

- 2nd generation (Hybrid): Gnutella 0.6
  Hierarchical layer (superpeers) with high degree, leaf nodes with low degree
  + As for Gnutella 0.4
  - As for Gnutella 0.4
  - Asymmetric load



1.2 Small World and Power Law Networks

1.2.1 Small World Networks

- Clustering coefficient c(v):
  $c(v) = \frac{e(v)}{deg(v)*(deg(v)-1)/2}$, where $e(v)$ denotes the number of connections between v's neighbors

1.2.2 Random Graphs

- Erdös-Renyi:
  - $g_{n,m}$ is a random chosen element from $G_{n,m}$ (set of all graphs with n nodes and m edges
  - $m \geq log(n) \implies$ connected component and diameter grows $log(n)$
- Gilbert
  - $g_{n,p}$ graph with n vertices
  - For each v,w draw an edge between them with probability p
  - Clustering coefficient asymptotically equal to p
- Watts-Strogatz-Model
  - Ring of n vertices
  - Rewire each edge with probability p to a random node $\implies$ „shortcuts“

1.2.3 Power-Law Distributed (scale-free)

- The probability a node ist connected to k nodes is $P(k)\ k^{-y}$ where $(2 < y \leq 3)$
  $\Longrightarrow$ Most vertices have a small degree, some „hubs“ have a high degree
  „The rich get richer“: new nodes will attach to a high degree node more likely

- Barbasi-Albert Model
  $\pi(v) = \frac{deg(v)}{\sum_{w \in V} deg(w)} \Longrightarrow$ like in **Gnutella**: nodes with high degree get more ping messages

- Copying Model
  - In each step copy a random node v an all it's connections
  - Connect v with v'

1.3 Structured P2P Approaches

- General
  - Location data not stored on the peer providing them but at other location in network
  - Responsibility is assigned by hash function, lookup directly from responsible peer
  - Common address space for data and nodes
  - Association may change (nodes enter/leave)
  - Search for data = routing to responsible node
  - Direct($\rightarrow$ small data stored in node) vs indirect($\rightarrow$ $(key, value)$ with value = pointer to download) storage
- Joining of a new node
  1. Calculate node id
  2. Contact arbitrary node in DHT
  3. Assignment of hash range
  4. Copy k/v-pairs of hash range (maybe redundant)
  5. Bind into environment
- Failure of a node
  1. Use of redundant k/v-pairs
  2. Use of redundant/alternate routing paths
  3. k/v-pair retrievable if at least one copy remains
- Departure of a node
  1. Partitioning of hash range to neighbors
  2. Copy k/v-pairs to neighbors
  3. Unbind from environment

1.3.1 Chord

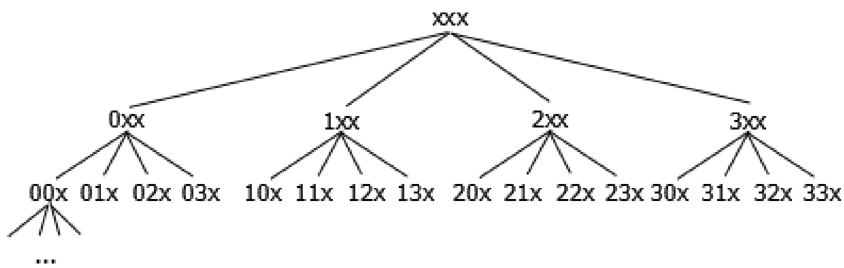
- General
  - $put(key, value)$ and $value = get(key)$
  - Ring topology with mix of short/long distance links
  - Finger table for node m

i	$m + 2^i$	succ.
0	$m + 1$	
1	$m + 2$	
2	$m + 4$	
3	$m + 8$	
...	...	...

  - Routing algorithm: To farthest finger predecessing k. On failure: Route to predecessor (do not overshoot!)
  - Soft-state approach: Delete k/v-pair after timeout
  - Store multiple successors, if succ[0] fails, take succ[1]
- Node join
  1. Pick ID (random, hash(IP+Port) or based on load balance or geographic position)
  2. Construct finger table (Query for each $m + 2^i$ successor and copy successor list from him)
  3. Update finger pointer to new node $\rightarrow \mathcal{O}(\log^2(n))$
- Routing not optimal in Chord (overlay vs. unterlay)

1.3.2 Pastry

- General
  - Prefix-based tree topology
  - $base = 2^b$
  - Leaves = key oder node ID
  - k/v-pair is managed by numerical closest node
  - 2^l-Bit identifiers ($i = 128$)



- Routing
  - Top-down in tree
  - Longest prefix match (1 prefix/step $\implies \mathcal{O}(\log_{2^b}(N)$
- Routing data per node
  1. Routing table (long distance links)
  2. Leaf set (numerically closest nodes)
  3. Neighbor set (closest nodes e.g. latency)

1. Routing table: $\lceil \log(N) \rceil$ rows with $2^b - 1$ entries each
   Example: $b = 2$, $N = 32$, $ID = 32101$

i / j	0	1	2	3
0	01230	13320	22222	-
1	30331	31230	-	33123*
2	...	-	...	...
3	-	...	...	...
4	...	-	...	...

   * All 33xyz are possible, choose topologically closest

2. Leaf set: Similar to Chord's successor list, fixed maximum size
   Node ID = 32101

Smaller Node-IDs		Higher Node-IDs	
32100	32023	...	...
32012	32022	...	...

3. Neighbor set: Fixed size, irrelevant for routing

- Routing with destination K at node N
  1. If K is in leaf set, route directly
  2. Determine common prefix (N, K)
  3. Search entry in routing table with longer prefix → route
  4. If not possible, search longest prefix from merged tables (routing, leafs, neighbors) → route (doesn't happen often)
- Node X wants to join Pastry DHT
  1. Determine node ID $(hash(IP : PORT))$
  2. Send JOIN to topologically nearest Pastry A_0
  3. Copy neighbor set from A_0
  4. A_0 routes JOIN to responsible node Z
     → Each node sends row in routing table → Missing entrys: Take IDs visited on route → Delete „own-ID-positions“
  5. Copy leaf set from Z
- Failure
  - „Are-you-alive“-messages
  - aks nodes from leaf set for their leaf set
  - aks neighbor in routing table for row
- Conclusion
  $\mathcal{O}(\log(n))$ hops, $\mathcal{O}(\log(n))$ storage
  → good support of locality

1.3.3 CAN

- General
  - D-dimensional value space
  - Complexity (search): $\mathcal{O}(\frac{D}{4} * N^{\frac{1}{D}})$
    $\frac{1}{2}$ because of the wrap-around
    $\frac{1}{2}$ because of average case
    $N^{\frac{1}{D}}$ hops in each dimension
  - Complexity (memory): $\mathcal{O}(D)$
  - „short-distance routing“
- Insertion of a new node
  1. Traverse tree until position is found
  2. „split“ partial tree
- Removal of a node
  1. Ideal case: Region can be merged
  2. Otherwise: Neighbor with smallest number of keys gets both (no merging!)
- Failure of node
  1. All neighbors start timer in proportion to size of region
  2. Smallest region/timer signals TAKEOVER first
- If N ist known before: Complexity routing and space $\mathcal{O}(\log(N))$
- Improvements:
  Multiple coordinate systems (with different hash functions) to archieve shorter paths (but r-time redundancy)
  Increase D to archieve more neighbors and thus shorter paths (but higher node state)

1.4 P2P Applications

1.4.1 Internet Indirection Infrastructure (i3)

- General
  - Framework on top of DHT
  - Allows multicast, anycast, mobility, QoS,... (what is to complex for network layer)
  - Association of data/services with ID → receiver(s) subscribe to content by ID („trigger“)
- i3 communication

Receiver: $insert(ID, R)$, R=IP:Port
Sender: $send(ID, data)$
Node resp. for ID: $send(R, data)$

- Mobility

ID	R_1

→

ID	R_2

- Multicast

ID	R_1
ID	R_2
ID	R_3

$\xrightarrow{scale}$

ID	R_1
ID	R_2
ID	X

and

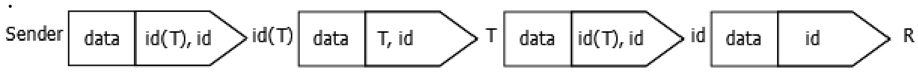
X	R_3
X	R_4

- Anycast

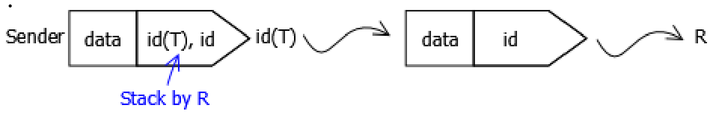
Prefix: ID of group/service
Postfix: Receiver selection by „longest prefix match“ (random, load balancing or geographical selection possible)

- Transcoder (in Stack of receivers)

Sender initiated :



Receiver initiated :



- Routing

Computer: Remove address from stack
Trigger: Replace ID with destination stack of trigger

- Triangle problem
  Solution: Choose close IDs for private communication (rendevouz point)
  How: Random choose and determine RTT

1.4.2 Bittorrent

- General
  - Disadvantage P2P: Huge files are downloaded from only one peer and uplink vs. downlink
  - General idea: Make use of idle uplink capacity of users
  - Split large files into chunks (chunks get IDs)
  - Parallel download: Load different chunks from different sources
- Components

Seeder: In Possession of the whole file

Leecher: Still needs chunk

Swarm: All peers sharing a file (torrent)

Tracker: Central registration instance
$\rightarrow$ Knows seeders and leechers $\rightarrow$ Coordinates communication between peers

- Torrent file
  - Provider hosts torrent file on a web server
  - Describes URL or tracker, file name, file size, chunk size, hash (integrity check)
- Chunk selection

Strict policy: Finish active chunks

Rarest first: Improve availibility of rare chunks

Random first chunk: Maybe the rare chunks are slow to get

Endgame Mode: Load last sub-chunk from multiple peers (fastest „wins“)

- Choking
  - Upload to peers who have uploaded to you recently
  - New peers are uploaded to on a trial basis
  - Optimistic unchoke:
    Rotate every 30 seconds, used to discover currently unused connections

- Anti-Snubbing
  - A peer finds itself beeing choked by all its peers (→ slow download)
  - Recover fast: 1 minute gone without receiving a sub-piece from X: Do not upload to it (except optimistic unchoke). → Instead use more optimistic unchokes to find new friends
- Upload-only Mode
  - After a download is ready, leecher becomes seeder
  - Upload to the peers with the best upload rate (fast replication)
- \+ scalability, high throughput
  \+ good fairness
  \- centralized tracker are easy to take down

Kapitel 2

Cloud Computing

2.1 Cloud

- Power consumption: 50% on idle, 90% on 50% utilized
- „Server“ = 1000’s of computers (data center)
- Cyclical demand curves (daily, weekly,...)
- Pay-as-you-go paradigm, automatically
- Computing IN the internet
- Only need to know the API, not the underlaying infrastructure

Private: Everything managed

Infrastructure: Databases, security and applications

Platform: Applications

Software: Usage only

2.2 Distributed Storage

- General
  - Key/value store
  - Similar to SQL
  - Scale up (boost ONE server) vs. scale out (buy more servers)
- CAP Theorem
  A system can only archieve 2 of this 3 things (Cassandra has 2+3):

Consistency: All nodes have the same data

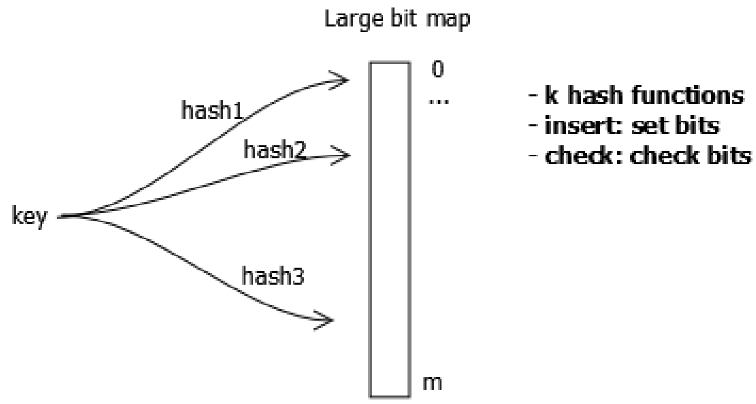
Availability: Allow all operation all the time

Partition-tolerance: Continue to work in spite of network partitions

2.2.1 Cassandra

- General
  - „NoSQL“, not only SQL (some columns are missing from some entries)
  - Put(key, value)
  - Get(value)
  - Often write-heavy
- Partitioning
  - Nodes logically structured in ring topology
  - Hashing
  - Lightly loeaded nodes move position to highly loaded nodes (balance)
- Replication
  - Each data item is replicated at N nodes
  - Rack unaware: Replicate at N-1 successors
  - Rack aware: Use a coordinator in rack level
  - Datacenter aware: Use a coordinator in datacenter level

- Write Operations
  1. Client issues write request at a random node
  2. Partitioner determines the node responsible for the data
  3. Log to disk commit log
  4. Modify memtable
  5. Flush memtable to disk (final/read-only sstable)
- Bloom Filter
  - Existence-check is cheap
  - False positives
  - Never false negatives



- Deletes
  Don't delete right away, but add tombstone
- Read
  Similar to write, except: Front-end node contacts closest replica, but also fetches data from multiple replicas (consistency)

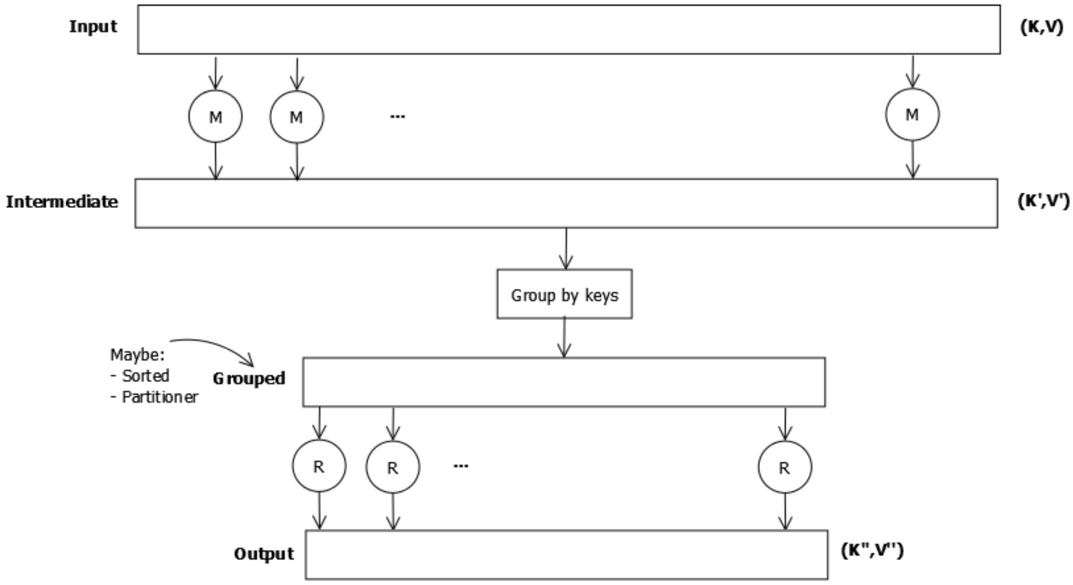
2.3 Distributed Data Processing

How to operate on distributed data? $\Longrightarrow$ Parallelize and process data directly at storage location

- Amdahl's Law
  - $S = \frac{1}{(1-p)+\frac{p}{n}}$, s=speedup, n=#processors, p=portion of program is parallelizable
  - Upper bound only (communication overhead)
- Request Level Parallelism (RLP)
  - Partition within a request AND across different requests
  - e.g. Google: Request → spell checker, ad server, index server,...
  - Redundant copies of indicies and documents (e.g. „super bowl 2013")
- Data Level Parallelism (DLP)
  - Processing large amount of raw data
  - Challenge: Parallelize computation, distribute data
  - → Map-Reduce

2.3.1 Map Reduce

- Master-Slave architecture: Slave pulls task from master
- Map
  Slice data into chunks:
  $map(in\_key, in\_value) \rightarrow list(out\_key, intermediatevalue)$
- Reduce
  - Collect and combine sub-problem solutions
  - $Reduce(out\_key, list(intermediatevalue)) \rightarrow list(out\_value)$



- Fault Tolerance
  - Restarting tasks
  - No heart beat $\implies$ Execute on a different TT
  - Locate slow tasks (Stragglers, speed < average -20%), run redundant $\implies$ take fastest („speculative execution“)

Kapitel 3

Anonymous Communication

Encryption protects only contents of communication, relationship between communicating parties remains visible.

3.1 Anonymous Communication

- Identify someone by:
  - IP
  - Browser Fingerprint
  - Search logs (e.g. AOL search engine)
- Censorship
  - **Public** (everyone knows something is blocked) vs. **silent** (noone knows the information exists)
  - Filtering based on:
    * Content (keywords)
    * Domain names / IP addresses
    * Author
    * User behavior (request history)
  - Blacklisting vs. whitelisting (more restrictive)
- Censorship techniques
  - Filtering URLs (proxy)
  - DNS censorship: Block domain names
  - Filtering of search results (Google China)
  - Exclusion from networks (servers and users)
  - Deletion of pieces of information (forum, database)

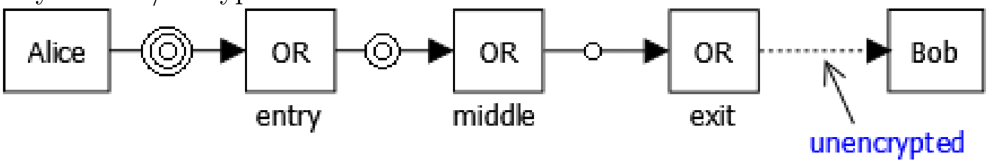
- Basic human right: Free speech, free information
- Identify someone

- Why use P2P Systems?
  - Server-based systems are not well suited
    - Manipulation is easy
    - Blocking a single server is easy
    - Trust?
  - P2P is able to
    - Store information redundantly
    - Retrieve information over multiple paths
    - These patzs are „black-box“-like
    - No central administration
- Communication types
  - High latency: Non interactive traffic, email
  - Low latency: Interactive traffic, instant messaging, TOR, JAP, I2P, ...

3.2 Techniques

3.2.1 TOR

- Clients select 3 onion routers (OR)
- Layered en-/decryption:



- Over 4000 nodes

3.2.2 Hidden Services in TOR

- Goal:
  - Deploy a server that anyone can connect to **without** knowing where it is or who runs it ($\Longrightarrow$ resistant to physical attacks)
  - Resistant to censorship
  - Can survice flood attacks
- Idea
  1. Server creates circuit to „introduction points“
  2. Server gives intro points addresses to service lookup directory
  3. Client obtains intro point address from directory
  4. Client creates circuit to a „rendevous point“
  5. Client sends address of rendevous point to server (through intro point)
  6. If server wants to talk to client: Connects to rendevous point
  7. Rendevous point mates the circuit from client & server
- Warning: Traffic between exit node and responder is **not** encrypted by TOR ($\rightarrow$ exit node can spy traffic)

3.2.3 Classification errors

Classification errors: false positives & false negatives

- A part c of all URLs are censored
- Classifier detects a censored URL with probability p and harmless with probability n correctly
- What is the probability a harmless connection is flagged as censored?
  $Pr(valid|alarm) = \frac{n \cdot (1-c)}{n \cdot (1-c) + p \cdot c}$
- Example: $c = 1\%, p = 90\%, n = 5\%$
  $\rightarrow$ Raised alarm is false with $\frac{0.05 \cdot 0.99}{0.05 \cdot 0.99 + 0.90 \cdot 0.01} = 85\%$!!

3.2.4 Crowds

Crowds: P2P system for protecting users' anonymity

- Crowd algorithm
  - Based on a simple randomized routing protocol
  - Each node runs a „jondo" process
  - Initiator always forwards a request to a random jondo
  - All later forward with probability p_f to another jondo, $1 - p_f$ to the end server (p_f is a system parameter): „coin toss"
- Analysing crowds:
  - p_f: Forward probability
  - n: # honest jondos
  - c: # colluding jondos ($n > c \geq 0$)
  - $\implies$ expected path length k:
    $P(x = k) = \underbrace{{p_f}^{k-2}}_{\text{steps to next jondos}} \cdot \underbrace{(1 - p_f)}_{\text{forward to end server}}$

3.2.5 Mixnets

- Idea
  - Send packet over several relays (mixes)
  - Each mix modifies (decrypts) the packet
  - Packet order is **not** kept
  - Padding: All packets have the same size
- Mix cascade:
  - Static mixing vs. dynamic mixing (user selects mixes)
  - User encrypts the packet with of each Mix: $E(E(E(...(E(msg, PK_n)..., PK_3)PK_2)PK_1)$
  - Each Mix: Decrypt and send to next destination
  - Last Mix: Deliver the packet
- Tasks of a Mix:
  - Decode messages (make packet recognition impossible)
  - Delete duplicates (prevent replay attacks)
  - Collect and **delay** messages (prevent temporal correlation)
  - Reorder messages (prevent temporal correlation)
- Types of Mixes
  - Threshold Mix: Buffer M packets, then flush all at once ($\implies$ Variable delay)
  - Timed Mix: Buffer packets for T seconds and flush all then ($\implies$ Fixed delay)
  - Threshold pool Mix: Buffer $M + F$ packets, flush only M random selected packets (F packets stay in buffer, potentially infinite delay)
  - Timed pool Mix: Empty buffer every T seconds but keep F randomly selected packets (only flush if more then F packets are in buffer, potentially infinite delay)

- Possible attacks
  - Trickle Attack (Timed Mixes): Block all traffic except the single target for T seconds
  - Flooding Attack (Threshold Mixes): Inject $N - 1$ packets after target Mix has flushed
  - Blending Attack (Pool Mixes): Combination of both
- Prevention against attacks
  - Encrypt packages between Mixes
  - Reroute traffic between Mixes dynamically
  - Alternate M and T (# packets to flush and time)
  - Send fake messages at flush

Kapitel 4

Sensor Networks

4.1 Sensor Networks Overview

- Possible applications for infrastructure-free networks:
  - Car-to-car communication
  - Military networking
  - Finding an empty parking lot (without servers)
- Challenges
  - No central entity → Participants must organize themselves (medium access, finding a route)
  - Limited range of wireless communication → Multi-hop network
  - Mobility
  - Battery-operated devices → Energy-efficient operation (e.g. low $\frac{\text{energy}}{\text{bit}}$)
- Wireless Sensor Networks (WSN)
  - Interacting with the environment instead of humans
  - MANET (Mobile ad-hoc network)
    Examples: Sensor on animals (rats), earthquake warning, volcanic activity, glacier (→ Mobile Internet Technology)
  - Here: WSN: Precision agriculture, logn-term surveillance of ill patients

  - Roles of participats in WSN:
    * **Sources** of data: Measure data with sensors
    * **Sinks** of data: Interested in receiving data (can be part of WSN or not)
    * Actuators: Control some device based on data (usually also a sink)
  - Deployment options for WNS
    * Random: Dropped by aircraft (uniform distributed)
    * Regular: Well planned, fixed
    * Mobile: Sensors can move to „interesting“ areas
  - Characteristic requirements for WNSs:
    * Quality of Service: No traditional QoS, but must still be „good“
    * Fault tolerance: Be robust against node failures (out of energy, destruction)
    * Lifetime: Network is important, individual nodes relatively unimportant
    * Scalability
    * Wide range of densities (small or vast numer of nodes per area?)
    * Programmability / flexibility of nodes
    * Maintainability: Self-monitoring
    * Reliability

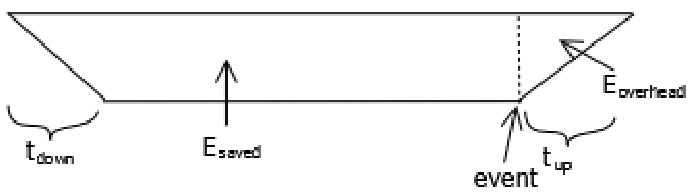
4.2 Node Architecture

- Main components of a WNS node:
  - Controller (μ-Controller)
  - Communication device(s): radio, light, ultrasound
  - Sensors / actuators
  - Memory
  - Power supply
- Transciever states
  - Transmit
  - Receive
  - Idle: Ready to receive
  - Sleep: Parts are switched off (recover time / startup energy?)

- Optical communication: Reflect laser by a mirror
- Ultra-wideband communication: Emit short „burst“ of power
  - Short pulse with large bandwidth
  - Requires tight time sync
  - Short range
  - Good wall penetration & multi-path propagation

4.2.1 Energy of WSN nodes

- Energy supply of mobile / sensor nodes
  - Primary batteries: Not rechargeable
  - Secondary batteries: Rechargeable by environment:
    * Light
    * Tempterature gradients
    * Vibrations
    * Pressure (e.g. on a shoe)
  - Energy consumption (example): $\frac{\text{Energy}}{\text{Instruction}} = 1nJ$,
    $Battery = 1J = 1Ws \implies 10^9$ instructions
- Switching between modes (active/sleep)
  1. Simple idea: Greedily switch to sleep wnenever possible
     - Problem: Time an power needed to switch to active mode again
     - Switching only pays off if $E_{\text{saved}} > E_{\text{overhead}}$



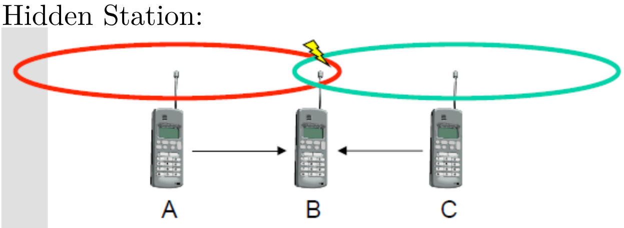
  2. Alternative: Dynamic voltage scaling
     - Run device with lower voltage & clock instead of changing modes
     - Power consumption p depends on clock frequency f and Voltage V:
       $P \sim f \cdot V^2$
- Time to transmit n Bits (R data rate, R_{code} coding rate):
  $\frac{n}{R \cdot R_{\text{code}}}$

- Computation vs. communication energy cost: Try to compute instead of communicate whenever possible!!!

4.3 Routing & Addressing

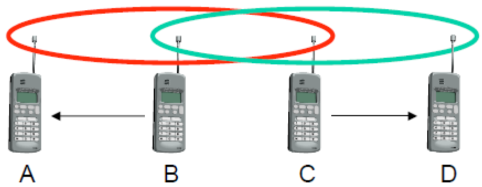
4.3.1 MAC

- Hidden Station:



  - C does not see A, sees a „free“ medium (CS fails)
  - A cannot hear the collision at B (CD fails)

- Exposed Station:



  - B sends to A, C has to wait (CS signals that the medium is in use)
  - But A is not in range of C $\Longrightarrow$ Waiting is not necessary!

- Receiving is about as expensive an transmitting
- Energy problems:
  - Collisions
  - Overhearing (Listen to a packet destinied for another node)
  - Idle listening (Listen when nobody is sending)
  - Protocol overhead

1. Centralized Medium Access (Polling, centralized computation of schedules

   - Simple, no collisions
   - Needs a central station
   - Overhead an delays
   - Big network sizes?

2. Contention-based protocols

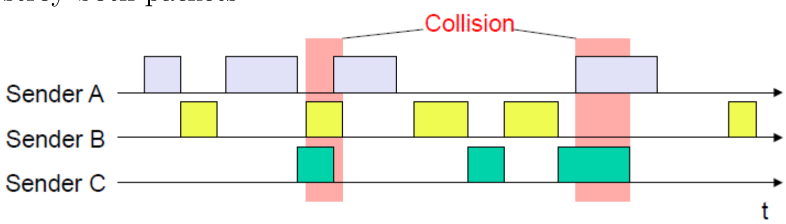
   - Risk of colliding packets is OK
   - Usually randomization somehow

3. Schedula-based MAC

   - A schedule exists (fixed or computed on demand)
   - Needed: Time syncronization!

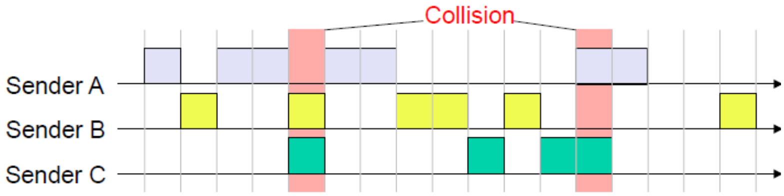
2. Contention-based Access (CSMA/CD Ethernet, Aloha, Slotted Aloha)

   - Protocoly specify: How to detect collisions and how o recover from collisions
   - Aloha
     - Idea: When you're ready: Transmit, detect collisions by ACK timeout
     - Recover from collision by retransmission after random interval
     - Problem: No common packet length, even small overlaps destroy both packets



     - Efficiency: 18%

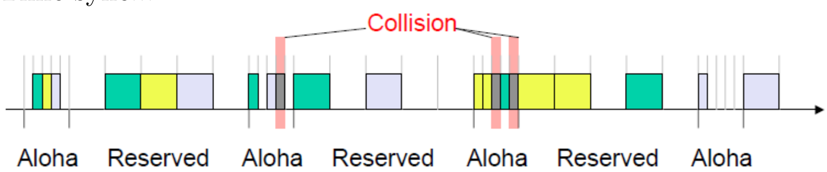
- Slotted Aloha
  - Sending must start at slot boundaries
  - Fixed packet length
  - If collision: Retransmit in future time slots with probability p, until success



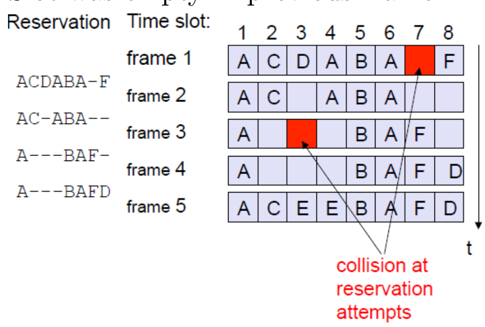
  - Effiency: 36%

3. Schedule-based MAC

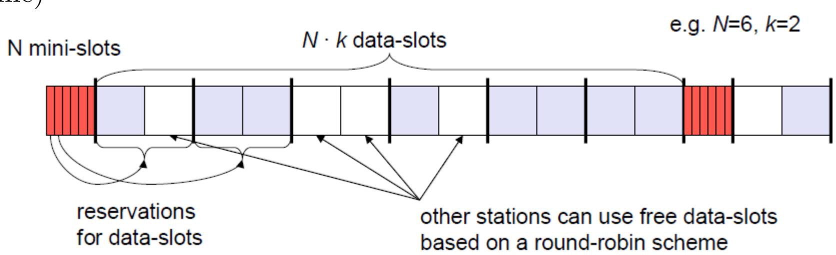
- DAMA (Demand Assigned Mutliple Access)
  - A sender reserves a future time slot
  - Sending within this time slot without collision
  - But: Higher delays due reservation
- Explicit Reservation (Reservation Aloha)
  - Aloha mode for reservation („competition“): Collision possible
  - Reserved mode: Transmission only withn reserved slots (no collisions)
  - Time sync!!!



- Implicit Reservation: PRMA (Packet Reservation MA)
  - Competition for empty slots (Slotted Aloha principle)
  - Reservation is valid until the station does **not** send in the reserved slot
  - Slot was empty in previous frame $\Longrightarrow$ New competition



- DAMA: Reservation-TDMA
  - Every frame consinsts of N mini-slots and x data-slots
  - Every station has its own mini-slot, can reserve up to k data-slots with this mini-slot ($x = N \cdot k$)
  - Unused data-slots: Other station can send (Round-Robin scheme)



4.3.2 Addressing

- In WSN: Often **content-based** addresses (But: Are not known before, have to be computed „in the field“)
- Names vs. addresses
  - Names: Refer to „things“, often unique
  - Addresses: Information needed to find these things, often unique
  - Name $\leftrightarrow$ Address: DNS, phonebook

- Distributed address assignmnet: Options
    1. Pick randomly addresses (risk of duplicates?)
    2. Avoid addresses used in local neighborhood (listen to channel before)
    3. Repair comflicts:
        - Pick random address
        - Send request
        - If address reply arrives, it already exists
    4. As 3, but contact a neighbor that has a fixed address already

4.3.3 Routing: Unicast id-cetric

- Each node has a unique ID
- „Routing“ = Construct a table that contains information how nodes can be reached
- „Forwarding“ = Use this table and forward to the next hop
- Optimization metric can be „smallest hop count“, „energy effiency“, „network lifetime“ (based of current battery levels), ...
- Ad-hoc routing protocols
    - Link state: Too much overhead
    - Distance vector: Too slow in reacting to changes
    - Simple solution: Flooding (simple but not acceptable in wireless systems because of energy waste & overhead)
- When does the routing protocol operate?
    1. Proactive: Always try to be up-to-date, have tables before they are actually needed
    2. Reactive: Determine route when actually needed (on demand)
    3. Hybrid: Combine 1+2

1. Proactive Routing: OLSR (Optimized Link State Routing)

   - LSR: Broadcast local link cost
   - Optimization:
     - X's broadcast is only forwarded by its multipoint relays
     - Multipoint relays: Set of X's neighbors which is connected to all two-step neighbors of X
     - $\implies$ Select a minimum (dominating set) of them

1. Proactive protocols: DSDV (Destination Sequence Distance Vector)

   - Add aging information to propagated route information (avoid loops)
   - Periodically send full route updates
   - On topology change, send **incremental** route updates ($\implies$ Unstable route updates are delayed)
   - Still lot of memory & traffic needed

2. Reactive protocols: DSR (Dynamic Source Routing)

1. phase: Flood the network (with a small discovery packet)
2. phase: Packet reaches destination
3. phase: Stored used path is send back as answer (along this path): Backward learning

2. Reactive protocols: AODC (Ad-hoc on Demand Vector)

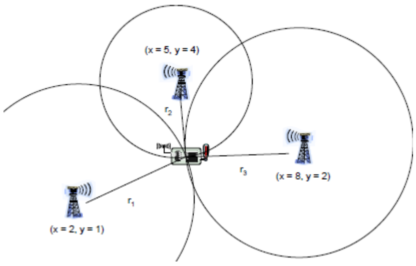
   - Very popular
   - Same as DSR, but nodes maintain routing tables instead of using Source Routing
   - Nodes on a route remember where packets came from $\rightarrow$ Routing tables
   - Less overhead but higher delay than proactive

- Link (Quality) Estimation
  - Which neighbors are good for communication?
  - ETX (Expected transmission count): Choose routes with high end-to-end throughput
  - # of data transmissions required to send a packet (including re-transmissions)
  - ETX of a route = sum of ETX of links on the route
  - Forward/reverse delivery ratio d_f/d_r: Probability that a data packet / ACK recieves
  - $ETX = \frac{1}{d_f \cdot d_r}$, ETX for each node to destination node X is sum of link ETX values
- Pro-active Routing: **Beacon** Vector Routing (BVR)
  - Virtual coordinate based addressing
  - Randomly select a few beacons
  - Construct trees from beacons to every other node
  - Every node knows its distance to every beacon (tree $\rightarrow$ reverse path)
    $\Longrightarrow$ This beacon vector = coordinates, e.g. $\langle q_1, q_2, ..., q_r \rangle = \langle 5, 1, ..., 3 \rangle$
  - Beacon Vector Routing in three parts:
    1. Greedy forwarding
    2. Fallback mode
    3. Scoped flooding
- 1. Main Rule: Minimize the sum of differences for the beacons that are closer to the destination than current p
     $\delta^+(p, d) = \sum_i max(p_i - d_i, 0)$, Rule: „Start - Dest."
  2. Ties in above $\rightarrow$ Minimize sum of differences to farther beacons: $\delta^-(p, d) = \sum_i max(d_i - p_i, 0)$, Rule: „Dest. - Start"
  3. If 1+2 Fail: Scoped flood

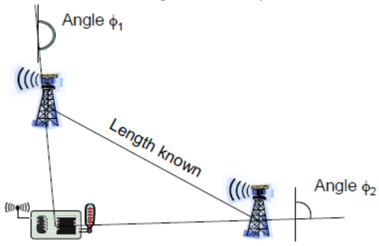
4.4 Localization

Localization: Need for a node to determine its **physical** position

- Proximity: Exploit finite range of wireless communication



- Trilateration / Multilateration and angulation: Use distance or angles with simple geometry



- Scene analysis: Measure environment „signatures" befordhand, then compare and create wireless fingerprints

- Recieved Signal Strengh Indicator (RSSI): Send out signal of fixed known strength to estimate distance
  Formula (not important):
  $P_{recv} = c\frac{P_{tx}}{d^{\alpha}} \Leftrightarrow d = \sqrt[\alpha]{\frac{c \cdot P_{tx}}{p_{recv}}}$

- Time of Arrival (ToA): Use time of transmission, propagation speed and time of arrival to compute distance. Exact time sync needed!

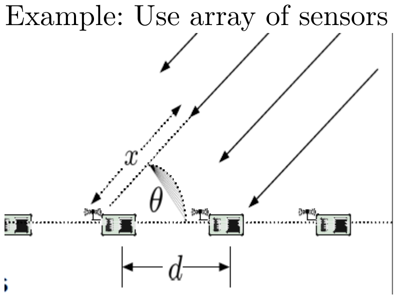
- Time Difference of Arrival (TDoA)
  - Use two different signals with different propagation speeds
  - Compute difference to compute distance
  - Problem: Calibration, expensive hardware, energy

- Also: Overlapping activity: Receiving mutliple signals $\Longrightarrow$ Must be in range of all of them

- Multihop range estimation (two ideas):
  1. Average hop length is known
     Distance = #hops · length per hop
  2. All exact hop lengths are known
     Distance = $\sum$ hop lengths
- Iterative multilateration: After a node calculated its location, share it with neighbors (Problem: Errors accumulate)

4.5 Time synchronization

- Example: Use array of sensors to estimate angle of arrival Θ



- Clocks in WSN nodes
  - Counter register is incremented by pulses
  - Register of node i at real time t is $H_i(t)$
  - Notation: Small letters = real time, capital letters = timestamp etc.
  - Clock speed drift: Θ_i = drift rate, Φ_i = phase shift
  - A node's software clock: $L_i(t) = \Theta_i \cdot H_i(t) + \Phi_i$
  - Tyme sync algorithms modify Θ_i and Φ_i, not the registers

4.5.1 External time sync

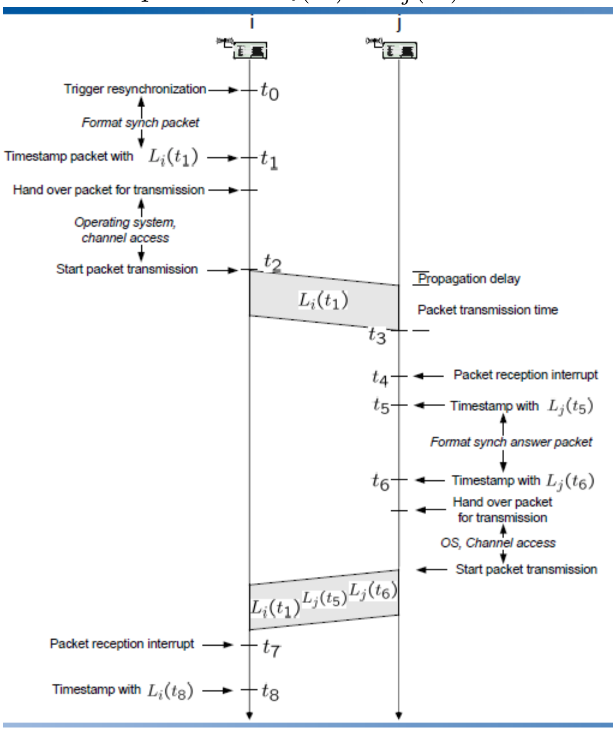
- Syc with external real time scale like UTC (atomic clock)
- At least one node must have access to external time scale

4.5.2 Internal time sync

No external timescale, nodes should have a „small time“ difference only (pairwise)

- Source of inaccuracies
  - Phases Φ_i are random (nodes are switched on at random times)
  - Oscullator deviation: ppm (pulse per million), depends on oszillator aging & envorinment
- Post-facto synchronization
  - Nodes do not sync all the time (energy)
  - External event happes at time t:
    * Node stores local timestamp t_i
    * Do time sync (neighbors / sink)
    * Convert $L_i(t)$ accodingly
- Time sync algorithms: Two fundamental classes
  1. Sender/reciever sync
  2. Reciever/reciever sync
- LTS (Lightweight Time Synchronization) ($\rightarrow$ image on the next page)
  - Sync all nodes to one reference clock
  - Correct only phase shofts, not drift rates
  - Pairwise sync, network-wide (minimum spanning tree, root = reference node R)
  - R syncs its neighbors, then first-level neightbors, ...
  - Cost per sync: 3 Packets $\rightarrow 3 \cdot n$ packets

- Goal: Compute $\Delta = L_i(t1) - L_j(t1)$



- Solution: $\Delta = \frac{L_i(t_8) - L_j(t_6)}{2} - \frac{L_j(t_5) - L_i(t_1)}{2}$

- Distributed Multihop LTS: No explicit construction of spanning tree, but implicit (node 42 syncs „directly“ with R)
- **Missing: TSync: Combines HRTS and ITR**

Kapitel 5

Internet of Things (IoT)

missing.

Kapitel 6

SDN

missing.