Final Exam Review
Papers to be covered


What is 6LoWPAN?

- IPv6 over Low-Power Wireless Area Networks
- Defined by IETF standards
  - RFC 4919, 4944
  - draft-ietf-6lowpan-hc and -nd
  - draft-ietf-roll-rpl
- Stateless header compression
- Enables a standard socket API
- Minimal use of code and memory
- Direct end-to-end Internet integration
  - Multiple topology options
Protocol Stack

TCP/IP Protocol Stack

- Application
- Transport
- Network
- Data Link
- Physical
- Ethernet PHY
- Ethernet MAC
- IP
- ICMP
- UDP
- TCP

6LoWPAN Protocol Stack

- Application
- Transport
- Network
- Data Link
- Physical
- IEEE 802.15.4 PHY
- IEEE 802.15.4 MAC
- IPv6 with LoWPAN
- UDP
- ICMP
Features

• Support for 64-bit and 16-bit 802.15.4 addressing
• Useful with low-power link layers such as IEEE 802.15.4, narrowband ISM and power-line communications
• Efficient header compression
  • IPv6 base and extension headers, UDP header
• Network autoconfiguration using neighbor discovery (ND)
• Unicast, multicast and broadcast support
  — Multicast is compressed and mapped to broadcast
• Fragmentation
  — 1280 byte IPv6 MTU -> 127 byte 802.15.4 frames
• Support for IP routing (e.g. IETF RPL)
6LoWPAN Architecture

- Single Edge Router
  - No route outside the LoWPAN

- Multiple Edge Routers with common backbone link
  - Backhaul link
  - Backbone link

- Simple LoWPAN
- Extended LoWPAN
- Ad-hoc LoWPAN
IPv4 vs. IPv6 Addressing

**An IPv4 address (dotted-decimal notation)**

172 . 16 . 254 . 1

10101100.00010000.11111110.00000001

One byte = Eight bits

Thirty-two bits (4 * 8), or 4 bytes

**An IPv6 address (in hexadecimal)**

2001:0DB8:AC10:FE01:0000:0000:0000:0000

Zeroes can be omitted

1000000000001:0000110110111000:10101100001000:1111111000000001:

000000000000000000000000000000000000000000000000000000000000000
IPv4 vs. IPv6 Header
6LoWPAN Stateless Header Compression

Non-Compressed fields...
\_ dispatch / \_ HC1 header/_

N.-C. fields...
\_ dispatch / \_ HC1 header/_ \_ HC2 header_/
Fragmentation

Initial 6LoWPAN fragment

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 1 1 0 0 0 |  datagram_size  |  datagram_tag  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  \  \ dispatch  _____
```

Non-initial 6LoWPAN fragment

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 1 1 1 0 0 |  datagram_size  |  datagram_tag  |
|  datagram_offset |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  \  \ dispatch  _____
```
WirelessHART Data Link Layer Design
Reliable Graph Routing

• Reliable Broadcast Graph \((G_B)\)
  - \(G_B\) is a graph connecting Gateway (GW) downward to all DEVs
  - Broadcasts common configuration and control messages
  - Each DEV has at least two parents in \(G_B\)

(a) Original network topology
(b) Broadcast graph
Reliable Graph Routing (Cont.)

• Reliable Uplink Graph ($G_U$)
  – $G_U$ is a graph connecting all DEVs upward to the Gateway
  – DEVs propagate periodic process data
  – Each DEV has at least two children in $G_U$
  – Both $G_B$ and $G_U$ have no fewer than 2 Access Points
Reliable Graph Routing (Cont.)

- Reliable Downlink Graph ($G_v$)
  - The graph from the Gateway to DEV $v$
  - Transmit unicast messages from the GW and NM to $v$
  - Each intermediate DEV has at least two children in $G_v$
  - There exists at least one directed cycle in $G_v$

(a) Original network topology

(b) Downlink graph to Dev 3 and Dev 4
Timing parameters of a job $J_j$

- Arrival time ($a_j$) or release time ($r_j$) is the time at which the job becomes ready for execution.
- Computation (execution) time ($C_j$) is the time necessary to the processor for executing the job without interruption.
- Absolute deadline ($d_j$) is the time at which the job should be completed.
- Relative deadline ($D_j$) is the time length between the arrival time and the absolute deadline.
- Start time ($s_j$) is the time at which the job starts its execution.
- Finishing time ($f_j$) is the time at which the job finishes its execution.
- Response time ($R_j$) is the time length at which the job finishes its execution after its arrival, which is $f_j - a_j$. 

![Diagram showing timing parameters](image)
Schedules for a set of jobs \{J_1, J_2, \ldots, J_N\}

- A schedule is an assignment of jobs to the processor, such that each job is executed until completion.
- A schedule can be defined as an integer step function \( \sigma : \mathbb{R} \rightarrow \mathbb{N} \), where \( \sigma(t) = j \) denotes job \( J_j \) is executed at time \( t \), and \( \sigma(t) = 0 \) denotes the system is idle at time \( t \).
- If \( \sigma(t) \) changes its value at some time \( t \), then the processor performs a context switch at time \( t \).
- Non-preemptive scheduling: there is only one interval with \( \sigma(t) = j \) for every \( J_j \), where \( t \) is covered by the interval.
- Preemptive scheduling: there could be more than one interval with \( \sigma(t) = j \).
Feasibility of Schedules and Schedulability

- A schedule is **feasible** if all jobs can be completed according to a set of specified constraints.
- A set of jobs is **schedulable** if there exists a feasible schedule for the set of jobs.
- A scheduling algorithm is **optimal** if it always produces a feasible schedule when one exists (under any scheduling algorithm).
Scheduling Algorithms

- Static Scheduling (offline, or clock-driven)
  - Static-Priority Scheduling
- Dynamic Scheduling (online, or priority-driven)
  - Dynamic-Priority Scheduling

- Preemptive vs. Non-preemptive
- Guarantee-Based vs. Best-Effort
- Optimal vs. Non-optimal
Metrics of Scheduling Algorithms (for Jobs)

Given a set $J$ of $n$ jobs, the common metrics are to minimize

- Average response time:
  \[ \sum_{J_j \in J} \frac{f_j - a_j}{|J|} \]

- Makespan (total completion time):
  \[ \max_{J_j \in J} f_j - \min_{J_j \in J} a_j \]

- Total weighted response time:
  \[ \sum_{J_j \in J} w_j (f_j - a_j) \]

- Maximum latency:
  \[ L_{\text{max}} = \max_{J_j \in J} (f_j - d_j) \]

- Number of late jobs:
  \[ N_{\text{late}} = \sum_{J_j \in J} \text{miss}(J_j), \]
  where $\text{miss}(J_j) = 0$ if $f_j \leq d_j$, and $\text{miss}(J_j) = 1$ otherwise.
Hard/Soft Real-Time Systems

- Hard Real-Time Systems
  - If any hard deadline is ever missed, then the system is incorrect
  - The tardiness for any job must be 0
  - **Examples**: Nuclear power plant control, flight control

- Soft Real-Time Systems
  - A soft deadline may *occasionally* be missed
  - Various definitions for “occasionally”
    - minimize the number of tardy jobs, minimize the maximum lateness, etc.
  - **Examples**: Telephone switches, multimedia applications

We mostly consider hard real-time systems in this course.
Recurrent Task Models

- When jobs (usually with the same computation requirement) are released recurrently, these jobs can be modeled by a recurrent task.

- **Periodic Task** $\tau_i$:
  - A job is released exactly and periodically by a period $T_i$.
  - A phase $\phi_i$ indicates when the first job is released.
  - A relative deadline $D_i$ for each job from task $\tau_i$.
  - $(\phi_i, C_i, T_i, D_i)$ is the specification of periodic task $\tau_i$, where $C_i$ is the worst-case execution time.

- **Sporadic Task** $\tau_i$:
  - $T_i$ is the minimal time between any two consecutive job releases.
  - A relative deadline $D_i$ for each job from task $\tau_i$.
  - $(C_i, T_i, D_i)$ is the specification of sporadic task $\tau_i$, where $C_i$ is the worst-case execution time.

- **Aperiodic Task**: Identical jobs released arbitrarily (we will revisit this part in Real-Time Calculus).
Evaluating A Schedule for Tasks

For a job $J_j$:
- Lateness $L_j$: delay of job completion with respect to its deadline.
  \[ L_j = f_j - d_j \]
- Tardiness $E_j$: the time that a job stays active after its deadline.
  \[ E_j = \max\{0, L_j\} \]
- Laxity (or Slack Time)($X_j$): The maximum time that a job can be delayed and still meet its deadline.
  \[ X_j = d_j - a_j - C_j \]

For a task $\tau_i$:
- Lateness $L_i$: maximum latency of jobs released by task $\tau_i$
- Tardiness $E_i$: maximum tardiness of jobs released by task $\tau_i$
- Laxity $X_i$: $D_i - C_i$;
Relative Deadline $\iff$ Period

For a task set, we say that the task set is with
- **implicit deadline** when the relative deadline $D_i$ is equal to the period $T_i$, i.e., $D_i = T_i$, for every task $\tau_i$,
- **constrained deadline** when the relative deadline $D_i$ is no more than the period $T_i$, i.e., $D_i \leq T_i$, for every task $\tau_i$, or
- **arbitrary deadline** when the relative deadline $D_i$ could be larger than the period $T_i$ for some task $\tau_i$. 
Some Definitions for Periodic Tasks

- The jobs of task $\tau_i$ are denoted $J_{i,1}, J_{i,2}, \ldots$.
- Synchronous system: Each task has a phase of 0.
- Asynchronous system: Phases are arbitrary.
- Hyperperiod: Least common multiple (LCM) of $T_i$.
- Task utilization of task $\tau_i$: $u_i = \frac{c_i}{T_i}$.
- System utilization: $\sum_{\tau_i} u_i$. 
Feasibility and Schedulability for Recurrent Tasks

- A schedule is **feasible** if all the jobs of all tasks can be completed according to a set of specified constraints.
- A set of tasks is **schedulable** if there exists a feasible schedule for the set of tasks.
- A scheduling algorithm is **optimal** if it always produces a feasible schedule when one exists (under any scheduling algorithm).
Comparison between RM and EDF (Implicit Deadlines)

**RM**
- Low run-time overhead: $O(1)$ with priority sorting in advance
- Optimal for static-priority
- Schedulability test is $NP$-hard (even if the relative deadline = period)
- Least upper bound: 0.693
- In general, more preemption

**EDF**
- High run-time overhead: $O(\log n)$ with balanced binary tree
- Optimal for dynamic-priority
- Schedulability test is easy (when the relative deadline = period)
- Least upper bound: 1
- In general, less preemption