Abstract:
• Generate a stateful schedule from an input language.
• Enforce deadlines in hard real-time systems via a scheduling algorithm that makes on-the-fly choices between communications to schedule on a network.
• Generate network code to execute schedules on real-time systems (RTLinux).

Background:
• Time Division Multiple Access (TDMA) is a common medium access protocol to implement real-time communication.
• TDMA provides guaranteed message delivery by assigning each process a timeslot during which it may communicate.
• If dependencies between communications are known, state information can be used to optimize the schedule.

Applications:
• Distributed hard real-time embedded systems found in the automotive, avionics, factory automation, and medical device domain.
• Particularly useful for fault-tolerant applications, because the dependency between the primary and the backups is known.

Technical Approach:

 Specifications:
• Specifies periodic processes, nodes, communications among them, and guards.
• Guards are defined to be functions executed by the scheduler that makes on-the-fly choices between communications at run-time.

 Scheduler:
• Generates schedules for the communication medium as well as CPU schedules for each node.
• Schedules represented as a graph.
• Processes scheduled up to the least common multiple of their periods.
• Network scheduler uses reverse earliest deadline first algorithm; CPU scheduler uses an earliest deadline first algorithm.

 Network Scheduling Algorithm:
• Initialize the current time to lcm (T1, T2, ..., Tn).
• While current time > 0:
  • Pick the active communication, C, whose sending task has the smallest slack.
  • If C is guarded
    • Schedule C as well as all of its alternate choices (specified by the guards) for the current timeslot
    • Else C is not guarded
    • Schedule C for the current timeslot
  • Decrement current time by one.
  • Update active tasks based on what was scheduled.

 Network Code:
• Assembly language-like instruction set consisting of the following:
c create (msgid, loc), destroy (msgid), future (dl, jmp), halt(), if (g, jmp), jmp (loc), receive (ch, loc), send (ch, msgid, relTime)
• This network code along with associated stubs and shared variable space layout runs on a network code machine, which executes on RTLinux.

 Network Code Example:
• Consider the following schedule, where a single communication (c1) is scheduled to transmit at time 9 from node 1 to node 2.

Scheduling Example:
• Consider the following input to our scheduler, which has 4 tasks running on 2 nodes. There are also 4 communications specified, and 2 guards which make on-the-fly decisions between the communications.

// Task section
// Task ID, Node, Period, WCET, Transmission Time
1, 1, 10, 3, 1
2, 1, 20, 2, 1
3, 2, 10, 5, 1
4, 2, 5, 1, 1

// Communication section
// Sending Task.Port -> Receiving Task.Port
1.1 -> 3.1
1.2 -> 4.1
2.1 -> 3.2
2.2 -> 4.2

// Guard section
// Guard ID, Function, Input Task, True Branch, False Branch
1, fun1, {2}, 1, 2
2, fun2, {1}, 3, 4

• The output from our scheduler is represented as a graph, which is then input to our visualization tool.

Conclusions:
• Created network scheduling algorithm which allows for on-the-fly choices.
• Developed a network code compiler.
• Implemented a visualization tool for graphic representation of schedules.