# PNETMAP: Virtual Network Implementation on a Partially-known Physical Network

Cristian Ferent
*Department of Electrical and Computer Engineering*
*State University of New York at Stony Brook*
*Stony Brook, NY, USA*
*cferent@ece.sunysb.edu*

Alex Doboli
*Department of Electrical and Computer Engineering*
*State University of New York at Stony Brook*
*Stony Brook, NY, USA*
*adoboli@ece.sunysb.edu*

***Abstract*—This paper presents PNETMAP, an algorithm that maps virtual communication networks of a Cyber Physical System onto a partially-known, physical network of reconfigurable embedded systems. The paper describes in detail the algorithm and offers an extensive set of experimental results on the quality of the produced communication structures for different network topologies. Experiments use a network of reconfigurable PSoC devices, from Cypress Semiconductor Inc.**

***Keywords*-cyber physical systems; virtual network; physical network; reconfigurable embedded systems;**

## I. Introduction

Cyber-Physical Systems (CPS) are envisioned to effectively integrate large-scale data acquisition, processing, and control to provide performance-efficient and dependable operation in dynamic conditions [1]. Data acquisition is over distributed physical areas, and uses a wide range of sensors that sense with various resolutions physical attributes, like temperature, humidity, gas composition, light intensity, and magnetic field. Processing includes local computing based on the sensed data as well as distributed algorithms for global decision making [2]. The embedded sensing nodes are connected through wired and/or wireless networks. CPS are essential for many modern applications, like urban transportation systems, intelligent power grid, health care, homeland security, and many more [1], [3].

CPS originate a new set of challenges, including dependable decision making for large, parallel and distributed embedded systems. Optimal decision making requires comprehensive knowledge of the input and state information over time [4]. This is hard to offer if data is acquired from physically distributed areas, pre-processed by resource-limited embedded nodes, and communicated over slow, unreliable communication links. Data aggregation helps efficient decision making under unreliable data acquisition conditions and tight resource and performance constraints as it produces more compact models [5] that simplify decision making and aid inferring properties of their behavior and performance (e.g., stability and latency). Data aggregation also helps compensate for noisy or missing data. Existing methods for wireless sensor networks (WSN) perform mostly single-level, monotonic aggregation based on basic functions like average, minimum and maximum [6]–[12]. Their goal is mainly to reduce the amount of transmitted data. In contrast, aggregation for CPS must produce precise models of the physical environment by combining data through non-monotonic functions, and organizing the models in semantic hierarchies based on the application specifics. Minimum-length communication topologies that are efficient for WSN might however contribute to creating imprecise models because valuable information is lost along the communication paths. For CPS, the communication paths can serve also as directions along which data aggregation for model construction is performed. In addition, various performance constraints must be satisfied regarding model resolution, timing, and event handling, in addition to tackling the more traditional metrics, like communication bandwidth and energy consumption. Thus, data communication for model construction through aggregation must be dependable under a wide range of operation conditions and constraints.

This paper presents PNETMAP, a data communication mapping algorithm for dependable, performance constrained CPS. PNETMAP is part of a design flow for a goal-oriented programming model [13], in which virtual spaces approximate the attributes of physical spaces. A small set of reference nodes are the link between virtual and physical spaces. As explained later in more detail, virtual spaces offer a good compromise between performance-predictable, dependable decision making and reactive response to dynamic conditions. PNETMAP maps a virtual network configuration for a virtual space onto a physical network of reconfigurable embedded nodes while preserving, as much as possible, the overall trace of the virtual paths and meeting the performance requirements of the paths, such as bandwidth and length. PNETMAP operates in two steps: first, an initial physical path is generated according to the virtual configuration, and second, the initial path is expanded by connecting neighboring nodes until a desired *path thickness* is achieved. The mapping algorithm is fully distributed. It

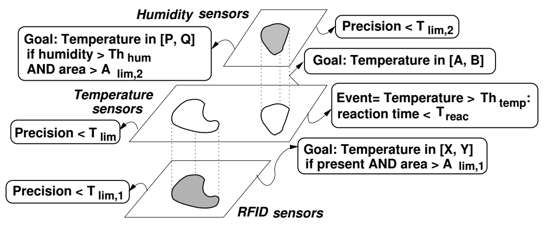

Humidity sensors
Goal: Temperature in [P, Q] if humidity > Th hum AND area > A lim,2
Precision < T lim,2
Goal: Temperature in [A, B]
Temperature sensors
Event= Temperature > Th temp: reaction time < T reac
Precision < T lim
Goal: Temperature in [X, Y] if present AND area > A lim,1
Precision < T lim,1
RFID sensors

Figure 1. Macro actuation by networked sensors

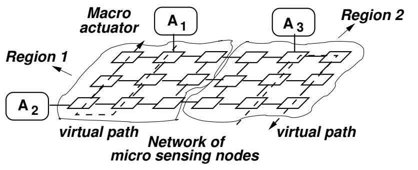


Figure 2. Virtual paths

performs only local decisions that involve a node and its closest neighbors. The two steps combine priority-based algorithms with backtracking.

The paper has the following structure. Section II offers an overview of the network application and the mapping algorithm. Section III gives a detailed presentation of PNETMAP. Considerations on communication interface, memory requirements and execution time are given in Section IV. Experimental results are presented in Section V. Finally, conclusions are put forth.

## II. Application environment and algorithm overview

Figure 1 shows the general characteristics of the tackled applications. Warehouse temperature monitoring is an illustrative example. Three layers of micro sensors are used to control macro actuators. The basic layer is a network of temperature sensors. The acquired information is used to maintain the temperature within the range $[A, B]$. An event occurs if in any point of the monitored region, temperature rises above the limit $C$. In this case, the reaction time for handling the event must be less than the timing constraint $D$. A second layer of humidity sensors monitors the air humidity, so that if humidity rises above a certain level then the temperature of that area is adjusted to the range $[P, Q]$. Finally, a third layer of RFID sensors is used to detect a certain kind of objects, so that the temperature of their storage space is set to the range $[X, Y]$. The application requires a data sampling precision, expressed in this case as the maximum time $T_{lim}$ between two samplings of the same temperature sensor.

The macro actuators are controlled based on data acquired through a network of distributed micro sensors, as shown in Figure 2. This paper considers that the application is executed on a grid of nodes connected through wired links. Every node is equipped with sensors. The node architecture is reconfigurable, thus the performance characteristics of the sampling modules, processing blocks, and communication modules can be changed dynamically. The control algorithms adapt dynamically the actuator behavior depending on the environmental characteristics (e.g., temperature, humidity, and position) and the activity of the other actuators. For example, the behavior of the control procedure of actuator $A_1$ in Figure 2 depends on the behavior of actuator $A_2$ as both actuators act on the same physical entities (specifically, the mass of air) inside the room. This control process requires having an accurate representation of the environmental characteristics over space and time.

The considered programming model [13] assumes two decision making spaces: (i) the virtual space and (ii) the physical space. The physical space is a representation of the real space characteristics, including static and moving objects, and environmental conditions. Every point in the space is described by a set of attributes, such as Cartesian position, temperature, humidity, gas composition, etc. The virtual space is an approximation of the physical space. Only a small subset of points (called *reference nodes*) are mapped from the physical space to the virtual space. They establish a one-to-one correspondence between the two spaces. Reference points have all attributes of the similar points in the physical space.

Using virtual spaces for description offers several advantages. They offer a good compromise between performance-predictable decision making and reactive response to dynamic conditions. Virtual spaces are used to compute long-term (strategic) decisions [13]. These decisions define constraints that must be constantly met by the short-term (tactical) decisions at the physical level. Decisions for virtual spaces are more predictable as they are computed using comprehensive information over time and space. This information captures general trends of the application, and thus changes less often. In contrast, decisions for physical spaces are more flexible to the specific conditions during operation but their overall performance is harder to estimate. Second, virtual spaces improve the scalability of the approach as global decisions do not have to consider detailed information besides that available at the reference points. Some of the physical information (e.g., global time and position of all sensors) is costly to obtain at the overall level. Having virtual spaces reduces the time and resource overhead of decision making as compared to solutions where data has to be constantly updated at the global level.

PNETMAP maps a virtual network configuration, represented by a number of virtual nodes arranged in a grid pattern and a set of paths defined within this virtual grid, onto a physical (real) network of embedded nodes. The algorithm explores the unknown network configuration between two such reference nodes to determine the physical-level paths while preserving, as much as possible, the overall trace of the virtual paths. For PNETMAP to generate a

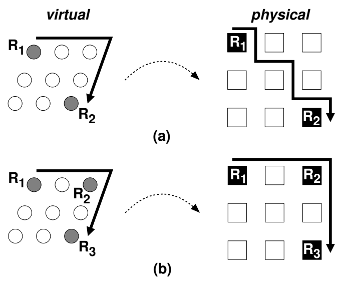


Figure 3. Virtual to physical mapping. (a) Virtual path with limited number of reference nodes; (b) Virtual path with reference nodes at each turning point

physical path closely matching a virtual path, it is desirable to build the virtual paths such that reference nodes are used wherever the virtual path changes direction. This is not a hard requirement, however, it helps in having a close resemblance between the virtual and physical paths. Figure 3 illustrates the difference between virtual and physical paths with and without this requirement. We call *virtual path segment* the part of a virtual path between two consecutive reference points. Virtual path segments are mapped to physical segments.

Each embedded node can have two different functionalities, sensing and aggregation. Hence, two types of nodes are present in a network: (i) data sensing nodes and (ii) data aggregation nodes. Data sensing nodes perform data acquisition using different sensor types and transmit this data throughout the network along the defined paths. In addition to data sensing functionality, aggregation nodes also perform data mining methods on both sensed and received data, while transmitting aggregation results along the physical paths. In the virtual space, all nodes implement the aggregation functionality. Since the physical network topology is expected to differ from the virtual grid, PNETMAP assigns real nodes as aggregation or sensing by using a uniform distribution of the virtual aggregation nodes on the corresponding path segments. This implies having the same number of aggregation points on both virtual and physical segments, with the physical segment having additional sensing points, if the length of the physical segment is greater than the length of the virtual segment.

The proposed mapping algorithm operates in two steps: first, an initial physical path is generated according to the virtual configuration and second, the initial path is expanded by connecting neighboring nodes to it until a desired *path thickness* is achieved. The thickness of the physical path is defined here as the grid distance between nodes added in the second step and the closest node added to the path in the first step. This two step approach is desirable with respect to the programming framework [13] in which PNETMAP is included. Apart from allowing decision making and data aggregation to cover a wider physical area, it also enables application specific reconfiguration of the physical paths. Integrating the two steps into a single flow is avoided due to the distribution of aggregation nodes along a physical path segment being known only when an entire segment is mapped while nodes added in the path expanding step are preferably linked to data sensing nodes which have a smaller computational load. Given that the mapping algorithm is fully distributed, PNETMAP performs only local decisions that involve the current node and its immediate neighbors.

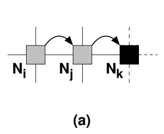
## III. Algorithm Details

PNETMAP operates on wired embedded systems networks. Both virtual and physical spaces have a grid network topology. For the physical grid, Cartesian coordinates define the position of a node with its neighbors being considered in one of the four directions: *up*, *left*, *down*, and *right*. We call physical networks with unavailable nodes (due to malfunction) *partial grid* networks and networks with all available nodes are referred to as *full grid* networks.

Before PNETMAP is executed, a network discovery process is performed by each available node in the physical grid. This determines the status of its neighbors (on-line or unavailable) along the four possible directions. An inquiry signal is transmitted and a reply is expected from online neighbors within a predefined time-out period. This functionality is not integrated in PNETMAP since in the goal-oriented programing scheme the network discovery process is required as a standalone primitive.

### A. Initial Path Generation

In step one (initial path generation), PNETMAP uses sequences of two consecutive reference nodes from the virtual path to find a similar path segment in the physical grid. Starting from the first reference node (segment start), the algorithm advances to any available neighbor in the direction imposed by the coordinates of the second reference node (segment end). Since the physical network is considered to have a structure close to a grid, an available direction is one for which the neighboring node is online or if it has not been previously explored. For each node, two constraints are checked before it is added to the physical path: (i) the total path latency constraint and (ii) the available data rate constraint. The latency constraint defines the maximum allowable length of the physical path. The data rate constraint is used to verify and set the communication interface rate and data sensing rate of the sensing nodes. Both latency and rate constraints are propagated along the mapped path as the algorithm advances. Once a physical segment is produced, the algorithm sets the participating nodes as either aggregation or sensing nodes based on the number of nodes on the corresponding virtual segment. The aggregation nodes are uniformly distributed along the physical path.



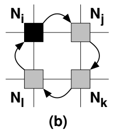
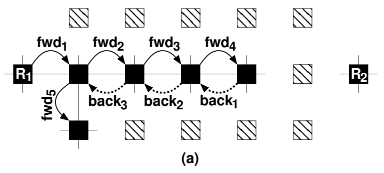


Figure 4. Scenarios which require backtracking. (a) No new direction available; (b) Path cycle generated. Black: node where backtracking is required

```
initialPathGeneration(vPath,ref. points){
  for each vPath segment
    currentNode=segmentStart
    while currentNode!=segmentEnd
      check directions:
      if unused direction exists
        select an unused available direction d
        if constraints(currentNode,d) are met
          add currentNode to pPath with direction d
          mark direction d as used
          currentNode=neighbor in direction d
        else
          mark direction d as used
          go to check directions
      else
        backtrack:
        currentNode=previousNode from pPath
        if currentNode=segmentStart & \
          no unused directions available
            return failed
    add segmentEnd to pPath
  return success
}
```

Figure 5. Initial path generation process pseudocode



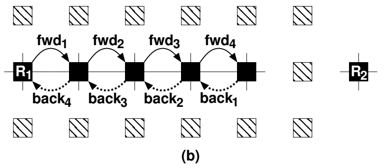


Figure 6. Physical network topology impact on backtracking. Black: on-line node, white hashed: unavailable node. (a) Backtracking length; (b) Backtracking to segment start with no other alternatives

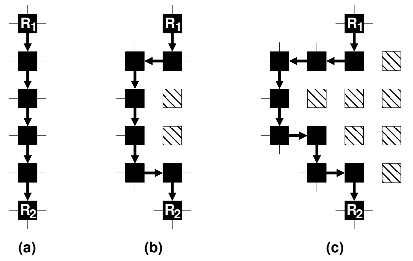


Figure 7. Physical network topology impact on path length. Black: on-line node, white hashed: unavailable node. (a) No unavailable nodes: segment length 6; (b) Unavailable nodes cluster increases segment length by 33%; (c) Unavailable nodes cluster increases segment length by 66%

There are situations that require the algorithm to backtrack and use a different set of physical nodes or directions to build an initial path segment. Such cases are encountered when, at the present node, some of the imposed constraints are not met, or its neighbors are not available, thus, rendering all directions unavailable. The algorithm must backtrack to select another set of alternatives that might satisfy the constraints or take a different route to avoid unavailable nodes. Backtracking is also needed when a cycle is generated on the initial path, such as when a node is reached that is already assigned to the same path. The two backtracking scenarios are shown in Figure 4. Before the algorithm backtracks, the directions that have been explored without success are marked as unavailable, therefore preventing subsequent explorations of the same unproductive alternative.

The pseudocode of the initial path generating step is shown in Figure 5. *vPath* is the virtual path, and *pPath* represents the generated physical path.

The physical topology characteristics, such as the number and position of the unavailable nodes, determines the length of the backtracking steps. Since the algorithm moves towards the segment end, a situation like in Figure 6(a) can be encountered. In this case, PNETMAP tries to determine a path segment defined by reference points $R_1$ and $R_2$. The algorithm advances by applying the forward steps 1, 2, 3, and 4. After step $fwd_4$, the current node does not allow the advancing in any direction (all neighbors unavailable), so the method backtracks three times, until another available and unexplored alternative is found to allow forward advance (such as $fwd_5$). Figure 6(a) shows how the topology parameters correlate to the length of backtracking. Note that backtracking can still fail if the network topology or the constraints force the algorithm to return to the segment start and no other unexplored alternatives are available. Figure 6(b) illustrates such a situation.

Clustering unavailable nodes affects the length of the physical paths. Three different situations are shown in Figure 7 where the lengths of the physical path segments are affected by the grid characteristics. Depending on the size and layout of the unavailable nodes cluster, the length of a segment can increase by more than 50%, as seen in Figure 7(c).

## B. Path Thickening

In step two (path thickening), PNETMAP performs path expansion for each previously generated path, by trying to attach neighbors to the path. This is a progressive process, applied from node to node, until the specified path thickness is obtained or the unavailable nodes prevent further attachments.

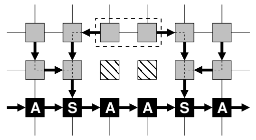


Figure 8. Initial path expansion process. Black: initial path node, gray: expansion added node, white hashed: unavailable node; A: aggregation, S: sensing

The path thickening routine traverses the initial path and builds for each node a list of the directions that are not already assigned. The algorithm processes neighbors only along these directions. For each such neighbor (called current node), a list of its neighbors is generated by inquiring all possible directions. Once this list is built, PNETMAP determines the best connection for the current node based on a set of priorities. A new node is preferably connected along the original expansion direction and to a data sensing node. If such a node is not found, the algorithm checks if any of the remaining neighbors of the new node are connected to an initial path data sensing node, and a connection to this node is produced. If the second priority fails, the new node is connected in the original expansion direction to a data aggregation node from the initial path. In the event that multiple neighbors satisfy the same priorities, the current node is connected to the neighbor with the least number of used connections and furthest down the path. The given set of priorities minimizes the length of parallel path routes, and ensures that new nodes are most likely connected to data sensing nodes. This is an advantage since it reduces the amount of data traffic through aggregation nodes, which are already involved in performing intensive computations for data mining.

Once a new node is added to the path, based on the previous priorities, the algorithm decides if any of the node's neighbors (which are already assigned to the current physical path) can be connected to the current node to improve the quality of the overall path mapping. If any neighbor matches the criteria, a signal is transmitted to inform the node to change its current path connection to the current node. Any neighbors that are not assigned to this path are instructed to connect to the current node, and hence get linked to the physical path. While most of the nodes set their connections based on the priorities of the expansion step, this action is required to identify neighbors that might be otherwise missed. Figure 8 shows such a case. If the unassigned neighboring nodes are not instructed to connect to the current node, the path thickening step would never reach them due to the specific layout of the unavailable nodes.

The same process is then applied to the next node along the original path expansion direction, until the set thickness is achieved. At this point, the algorithm returns to the initial

```
expandPath(vPath,pPath,thickness){
  for each node on initial  pPath
    for each direction not used on pPath
      expandDirection=currentDirection
      while pPath thickness not obtained & \
        expandDirection available
          currentNode=neighbor in expandDirection
            using priority list and constraints
              add/reconnect currentNode to pPath
            for all other neighbors of currentNode
              if neighbor is on pPath
                using priority list and constraints
                  check reconnect to currentNode
              else
                connect neighbor to currentNode
      return to initial pPath node
}
```

Figure 9. Path thickening process pseudocode

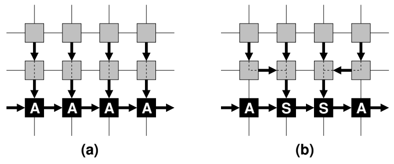


Figure 10. Impact of aggregation/sensing nodes distribution on generated path. Black: initial path node, gray: expansion added node; A: aggregation, S: sensing. (a) Distribution generated by 4 virtual nodes for 4 physical nodes; (b) Distribution generated by 2 virtual nodes for 4 physical nodes

path, and the next node is analyzed until all nodes are investigated. Before adding a new node to the path, the latency and rate constraints are checked.

The pseudocode of path thickening is presented in Figure 9. A partial PNETMAP result is shown in Figure 10 for different mappings of virtual grids onto the same physical network. The number of virtual nodes imposes a different distribution of the aggregation and sensing nodes in the physical network. The generated results are different since the priority list used in the thickening process defines a connection to a data sensing node. The execution times for the two scenarios in Figure 10 also show small variations due to the signals that are transmitted in each case.

In contrast to the initial path generation step, path thickening does not backtrack. This is not a limitation since, based on the priority list, every node and every neighbor is connected such that the overall result is optimized. This includes reconnecting nodes to different points, if a higher priority is satisfied as thickening advances. In fact, this is similar to backtracking, with the range limited to the immediate neighbors. As shown in the fallowing sections, backtracking lengths of one and two solve most situations. Using a neighbor window for path thickening is similar to a maximum backtracking length of two. Hence, path thickening does not require backtracking.

## IV. Communication, Memory and Execution Time Considerations

The considered physical grid network is based on the embedded PSoC processor from Cypress Semiconductor Inc. [14]. The communication links in each direction are implemented using the UART module of PSoC. PNETMAP uses five basic communication packet types for implementing its functionality: (i) Advance packets contain the coordinates of the segment end reference point and information about path constraints. (ii) Backtrack packets inform the receiving node that a backtracking situation was discovered. (iii) Inquire neighbor packets contain the path ID for which information is requested. (iv) Reply to inquiry packets tell the receiving node if the transmitting node is assigned or not to the inquired path as an aggregation or sensing node. They contain the total number of connections that the node is using for the inquired path. (v) Connect packets force the receiving node to establish a path link to the transmitting node. Minimizing the communication between nodes (e.g., the total number of bytes used by each packet) is done using bit encoding. This is needed since UART supports only byte-sized data transmissions.

The memory requirements of PNETMAP have been determined by considering that the memory word is one byte long. The algorithm uses the following data structure stored locally by a node: (i) node coordinates - 2 bytes; (ii) coordinates of reference node (segment end) - 2 bytes; (iii) list of available connections (directions) - 4 bytes for a full grid; (iv) for each available connection, a list of the paths that use the connection - 1 byte for each path; and (v) for backtracking, a flag is saved for each available connection to signal if it has been explored or not - 4 bytes for a full grid. In addition, 4 memory bytes are needed by the expansion process for storing information about each neighbor. Given a maximum number of five paths in the physical grid network, 36 bytes are required by the algorithm at each node. This memory requirement does not include any auxiliary locations needed to implement the algorithm's functionality.

The execution time for PNETMAP on a real PSoC network was measured with respect to the number of data bytes that had to be transmitted by the implementation. The data bytes were transmitted through the UART interface configured at a rate of 60,000 bps. The resulting time is converted to equivalent clock cycles for a PSoC device running at 24 MHz. For this configuration, 0.18 milliseconds are needed to transmit one byte. The time is equivalent to 4,320 clock cycles.

## V. Experiments

All experiments described in this section have been performed on a set of three virtual paths, shown in Figure 11. $Path_1$ and $Path_3$ bound the different virtual grid configu-

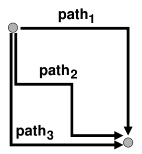


Figure 11. Virtual paths configuration used in experiments

rations considered. This includes different numbers of total virtual nodes and different numbers of reference points.

Experimental results report only the time for transmitting the required packets through the physical grid. The execution time of the PNETMAP algorithm on a node is not included. For the used transmission rate, this approximation is accurate enough because the algorithm uses only a few *if* statements. Implemented efficiently in assembly language, the execution time of PNETMAP is only a few hundreds of clock cycles, which is insignificant as compared to the thousands of clock cycles required to transmit one byte at 60,000 bps. Due to the same reason, the delay between a node receiving a packet and sending back a reply was also ignored. This approximation produces a significant error only for higher baud rates. Results of the execution times are presented in both milliseconds and equivalent clock cycles.

Randomly generated configurations have been used placing unavailable nodes in the physical grid layout. The network discovery process is performed before the algorithm is executed in order to identify the unavailable nodes. The process requires the available nodes to send one byte to all neighbors. Receiving a reply byte within a time-out period marks the neighboring node as on-line. Otherwise, a neighbor is unavailable. The considered time-out period is the time needed to transmit three bytes. The execution time of the discovery process is presented in Table I. As the number of unavailable nodes increases, the number of time-outs is also higher. However, this does not increase the execution time since the number of inquiries and replies reduces, thus causing a decrease in the overall execution time.

### A. Execution Time

This section discusses the execution time of PNETMAP. Different scenarios have been investigated, including different virtual path thickness and various network configurations for both virtual and physical spaces.

In Table II, the time required to implement a virtual paths mapping is given for a full and partial grid physical network (10×10 configuration) for different numbers of virtual to real reference points. The path thickness set for the expansion process is ±2 and the virtual topology used is a 5×5 grid. Experiments suggest that the number of reference points does not impact the execution time of the initial path generation step for neither full nor partial grids. For path thickening, small variations of the execution

Table I
EXECUTION TIME FOR NETWORK DISCOVERY PROCESS

| Total # Nodes | # Off-line Nodes | # Inquiries | # Replies | # Time-outs | Exec. Time [ms] | Exec. Time [clk cyc] |
|---|---|---|---|---|---|---|
| 36 | 0 | 144 | 120 | 24 | 60.48 | 1,451,520 |
| 36 | 3 | 132 | 98 | 34 | 59.76 | 1,434,240 |
| 36 | 7 | 116 | 72 | 44 | 57.6 | 1,382,400 |
| 36 | 14 | 88 | 44 | 44 | 47.52 | 1,140,480 |
| 81 | 0 | 324 | 288 | 36 | 129.6 | 3,110,400 |
| 81 | 8 | 292 | 230 | 62 | 127.44 | 3,058,560 |
| 81 | 16 | 260 | 178 | 82 | 123.12 | 2,954,880 |
| 81 | 32 | 200 | 112 | 88 | 103.68 | 2,488,320 |
| 225 | 0 | 900 | 840 | 60 | 345.6 | 8,294,400 |
| 225 | 22 | 812 | 684 | 128 | 338.4 | 8,121,600 |
| 225 | 45 | 732 | 552 | 180 | 328.32 | 7,879,680 |
| 225 | 90 | 552 | 332 | 220 | 277.92 | 6,670,080 |

time exist with the change in number of reference points. This is due to the different aggregation and sensing nodes distributions generated. Situations similar to the ones shown in Figure 10 were encountered. The same argument is consistent for changes in the virtual space architecture. A maximum variation of 5% in execution time was observed for virtual networks of different sizes.

The execution times for different physical networks are shown in Tables III (full grid) and IV (partial grid). A 4×4 virtual network and multiple path thickness settings were used. Comparing the execution time needed for determining the initial path and the execution time required to expand the initial path, more than 50% of the total execution time is used by the second step. For larger physical networks, this percentage is much higher. Considering different levels of path thickness for the same number of available nodes, the execution time drastically increases as the thickness increases since more neighbors are explored.

The initial path generation execution time is almost linearly related to the number of nodes for full grid networks. For partial grid networks, the initial process requires more time, due to the increased path length caused by unavailable nodes distributions. For an 81-node, 20% partial grid network, a spike in the execution time can be observed. This is due to the random distribution of unavailable nodes creating a much longer initial path than for the same full grid network topology. The path thickening process execution time does not include such “spikes”. The time is almost linearly related to the total number of nodes for both full and partial grid networks. However, the expansion process requires less time as more nodes are unavailable, in contrast to the initial path generation process. This is because there are fewer neighbors that need to be explored by the algorithm.

The experimental results suggest that the critical step is the path thickening step. Even for an extreme number of 14 backtracking steps, the ratio between the initial and thickening steps does not go above 70%. Such high ratios are only observed for relatively small networks. At the other extreme, this ratio can be as low as 7% for large full grid networks and a path thickness of $\pm 4$ nodes.

PNETMAP fails in the initial step for two of the tested scenarios. This is due to the randomly generated unavailable nodes creating a topology for which PNETMAP does not find a solution after exploring all possible alternatives for a path segment. The topologies are similar to the one in Figure 6(b). To reduce the likelihood of such scenarios, PNETMAP has to eliminate the reference point (segment start) from the physical path and search for alternatives that skip this reference point. One possible solution would be to consider the last node added before blocking occurs as a new reference for defining a path segment, and then attempting to map this configuration.

### B. *Longest and Shortest Paths*

The lengths of the shortest and longest physical paths generated by PNETMAP for full grid and partial grid networks are shown in Table V and Table VI. These paths were determined based on the virtual paths in Figure 11. The shortest path is considered as the length of the initial path only, while the longest path also includes the thickening process.

For full grid networks (Table V), the shortest path length is equal to the grid distance required to implement the path. The longest path is imposed by the path thickness. Due to the nature of the expansion process, an absolute maximum longest path that can be generated by the algorithm is equal to the length of initial path + path thickness + one node. The worst case is reached in few situations, since the aggregation and sensing nodes distribution dictates the connections selected by the thickening step, as nodes are attached further down the path, if possible.

Table II
EXECUTION TIME FOR DIFFERENT NUMBERS OF REFERENCE POINTS ($5 \times 5$ VIRTUAL NETWORK AND $10 \times 10$ PHYSICAL NETWORK)

| **# Ref. Nodes** | **# Off-line Nodes** | **Total # Back Steps** | **Initial Path Exec. Time [ms]** | **Path Expansion Exec. Time [ms]** | **Total Exec. Time [ms]** | **Total Exec. Time [clk cyc]** |
|---|---|---|---|---|---|---|
| 7 | 0 | 0 | 29.7 | 200.88 | 230.58 | 5,533,920 |
| 7 | 10 | 2 | 35.1 | 182.7 | 217.8 | 5,227,200 |
| 7 | 20 | 1 | 36.72 | 124.92 | 161.64 | 3,879,360 |
| 11 | 0 | 0 | 29.7 | 200.88 | 230.58 | 5,533,920 |
| 11 | 10 | 2 | 35.1 | 181.98 | 217.08 | 5,209,920 |
| 11 | 20 | 1 | 36.72 | 124.38 | 161.1 | 3,866,400 |
| 15 | 0 | 0 | 29.7 | 200.88 | 230.58 | 5,533,920 |
| 15 | 10 | 2 | 35.1 | 180.18 | 215.28 | 5,166,720 |
| 15 | 20 | 1 | 36.72 | 124.38 | 161.1 | 3,866,400 |

Table III
EXECUTION TIME FOR FULL-GRID NETWORKS BASED ON A $4 \times 4$ VIRTUAL NETWORK

| **Total # Nodes** | **Path Thickness** | **Initial Path Exec. Time [ms]** | **Path Expansion Exec. Time [ms]** | **Total Exec. Time [ms]** | **Total Exec. Time [clk cyc]** |
|---|---|---|---|---|---|
| 36 | $\pm 1$ | 16.74 | 58.68 | 75.42 | 1,810,080 |
| 36 | $\pm 2$ | 16.74 | 109.44 | 126.18 | 3,028,320 |
| 36 | $\pm 4$ | 16.74 | 186.3 | 203.04 | 4,872,960 |
| 49 | $\pm 1$ | 19.98 | 70.92 | 90.9 | 2,181,600 |
| 49 | $\pm 2$ | 19.98 | 133.02 | 153 | 3,672,000 |
| 49 | $\pm 4$ | 19.98 | 235.62 | 255.6 | 6,134,400 |
| 81 | $\pm 1$ | 26.46 | 95.76 | 122.22 | 2,933,280 |
| 81 | $\pm 2$ | 26.46 | 180.36 | 206.82 | 4,963,680 |
| 81 | $\pm 4$ | 26.46 | 344.16 | 370.62 | 8,894,880 |
| 100 | $\pm 1$ | 29.7 | 106.56 | 136.26 | 3,270,240 |
| 100 | $\pm 2$ | 29.7 | 200.88 | 230.58 | 5,533,920 |
| 100 | $\pm 4$ | 29.7 | 384.12 | 413.82 | 9,931,680 |
| 225 | $\pm 1$ | 45.9 | 170.28 | 216.18 | 5,188,320 |
| 225 | $\pm 2$ | 45.9 | 322.92 | 368.82 | 8,851,680 |
| 225 | $\pm 4$ | 45.9 | 622.8 | 668.7 | 16,048,800 |

For partial grid networks (Table VI), the length of both shortest and longest path depends on the distribution of unavailable nodes. There are cases with an increase in shortest path length of up to 106% for 14 backtracking steps. This was obtained for an 81-node, 20% partial grid network. For the same case, a spike in the initial step execution time was also observed. For the longest path, the same dependency on the path thickness and distribution of aggregation and sensing nodes is valid as for the full grid. In addition, the distribution of the unavailable nodes also influences the length by reducing the number of nodes that are added to the path during the thickening step.

### C. Algorithm Backtracking

A set of 150 experiments were performed on different partial grid networks with randomly generated unavailable nodes in an effort to establish a distribution of the lengths of backtracking and the directions that resolve a backtracking situation.

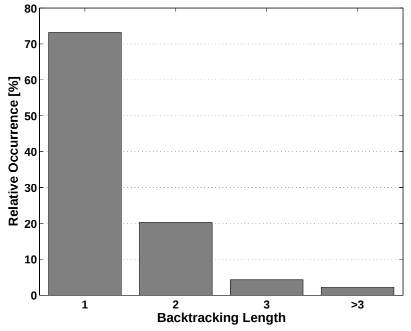


Figure 12. Relative overall backtracking length

Figure 12 presents the observed backtracking length distribution. Clearly, the dominant backtracking length used is one, fallowed by length two. These two cases account for

Table IV
EXECUTION TIME FOR PARTIAL-GRID NETWORKS BASED ON A 4 × 4 VIRTUAL NETWORK

| Total # Nodes | # Off-line Nodes | Path Thickness | Total # Back Steps | Initial Path Exec. Time [ms] | Path Expansion Exec. Time [ms] | Total Exec. Time [ms] | Total Exec. Time [clk cyc] |
|---|---|---|---|---|---|---|---|
| 36 | 3 | ±2 | 1 | 19.44 | 86.94 | 106.38 | 2,553,120 |
| 36 | 3 | ±4 | 1 | 19.44 | 128.52 | 147.96 | 3,551,040 |
| 36 | 7 | ±2 | 0 | 18.9 | 45 | 63.9 | 1,533,600 |
| 36 | 7 | ±4 | 0 | 18.9 | 59.94 | 78.84 | 1,892,160 |
| 36 | 14 | ±2 | FAIL | - | - | - | - |
| 36 | 14 | ±4 | FAIL | - | - | - | - |
| 81 | 8 | ±2 | 3 | 31.32 | 155.34 | 186.66 | 4,479,840 |
| 81 | 8 | ±4 | 3 | 31.32 | 261.54 | 292.86 | 7,028,640 |
| 81 | 16 | ±2 | 14 | 45.9 | 122.94 | 168.84 | 4,052,160 |
| 81 | 16 | ±4 | 14 | 45.9 | 175.32 | 221.22 | 5,309,280 |
| 81 | 32 | ±2 | 6 | 36.18 | 53.46 | 89.64 | 2,151,360 |
| 81 | 32 | ±4 | 6 | 36.18 | 79.42 | 115.6 | 2,774,400 |
| 225 | 22 | ±2 | 5 | 55.08 | 312.12 | 367.2 | 8,812,800 |
| 225 | 22 | ±4 | 5 | 55.08 | 523.08 | 578.16 | 13,875,840 |
| 225 | 45 | ±2 | 7 | 59.4 | 264.42 | 323.82 | 7,771,680 |
| 225 | 45 | ±4 | 7 | 59.4 | 385.74 | 445.14 | 10,683,360 |
| 225 | 90 | ±2 | FAIL | - | - | - | - |
| 225 | 90 | ±4 | FAIL | - | - | - | - |

Table V
PATH LENGTHS FOR FULL-GRID NETWORKS BASED ON A 4 × 4 VIRTUAL NETWORK

| Total # Nodes | Path Thickness | Shortest Path | Longest Path |
|---|---|---|---|
| 36 | ±1 | 11;11;11 | 12;11;12 |
| 36 | ±2 | 11;11;11 | 13;12;13 |
| 36 | ±4 | 11;11;11 | 15;13;13 |
| 49 | ±1 | 13;13;13 | 14;14;14 |
| 49 | ±2 | 13;13;13 | 15;15;15 |
| 49 | ±4 | 13;13;13 | 17;13;17 |
| 81 | ±1 | 17;17;17 | 18;18;18 |
| 81 | ±2 | 17;17;17 | 19;19;19 |
| 81 | ±4 | 17;17;17 | 21;17;21 |
| 100 | ±1 | 19;19;19 | 20;20;20 |
| 100 | ±2 | 19;19;19 | 21;21;21 |
| 100 | ±4 | 19;19;19 | 23;21;23 |
| 225 | ±1 | 29;29;29 | 30;30;30 |
| 225 | ±2 | 29;29;29 | 31;31;31 |
| 225 | ±4 | 29;29;29 | 33;33;33 |

Table VI
PATH LENGTHS FOR PARTIAL-GRID NETWORKS BASED ON A 4 × 4 VIRTUAL NETWORK

| Total # Nodes | # Off-line Nodes | Path Thickness | Shortest Path | Longest Path |
|---|---|---|---|---|
| 36 | 3 | ±2 | 15;11;11 | 16;12;12 |
| 36 | 3 | ±4 | 15;11;11 | 16;12;12 |
| 36 | 7 | ±2 | 15;11;11 | 16;11;12 |
| 36 | 7 | ±4 | 15;11;11 | 16;11;11 |
| 36 | 14 | ±2 | FAIL | FAIL |
| 36 | 14 | ±4 | FAIL | FAIL |
| 81 | 8 | ±2 | 23;17;17 | 24;19;19 |
| 81 | 8 | ±4 | 23;17;17 | 25;19;21 |
| 81 | 16 | ±2 | 35;21;17 | 36;21;19 |
| 81 | 16 | ±4 | 35;21;17 | 37;21;22 |
| 81 | 32 | ±2 | 25;21;17 | 29;25;19 |
| 81 | 32 | ±4 | 25;21;17 | 29;25;21 |
| 225 | 22 | ±2 | 39;31;29 | 39;33;31 |
| 225 | 22 | ±4 | 39;31;29 | 40;35;33 |
| 225 | 45 | ±2 | 45;31;29 | 46;33;31 |
| 225 | 45 | ±4 | 45;31;29 | 48;35;33 |
| 225 | 90 | ±2 | FAIL | FAIL |
| 225 | 90 | ±4 | FAIL | FAIL |

more than 90% of the total 560 backtracking situations. The average number of backtracking locations per test case was four. Assuming a worst case scenario of only length three solving four such cases, an increase in execution time of only 6 ms results. This is less than 10% of the smallest total execution time. Further improving the backtracking of PNETMAP does not significantly reduce the execution time due to the dominance of the path thickening step on the total execution time, especially for large networks.

The distribution of directions which solve the backtracking situations is shown in Figure 13. The close resemblance of this distribution to a uniform distribution proves that using a preference-based selection of directions is not ad-

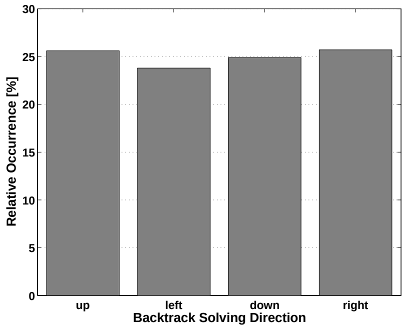


Figure 13. Relative overall backtrack solving direction

visable for backtracking. The added complexity of such an implementation does not compensate the small, if any at all, improvement in execution time.

## VI. Conclusion

This paper presents PNETMAP, a data communication mapping algorithm for dependable, performance constrained CPS. PNETMAP is part of a design flow for goal-oriented programming, in which virtual spaces approximate the attributes of physical spaces. PNETMAP maps a virtual network structure for a virtual space onto a physical network of embedded nodes while preserving, as much as possible, the overall trace of the virtual paths and meeting the performance requirements of the paths, such as bandwidth and length. PNETMAP executes two steps, first, an initial physical path is generated according to the virtual configuration, and second, the initial path is expanded by connecting neighboring nodes.

Experiments performed on a network of reconfigurable PSoC devices running at 24 MHz indicate that the execution time of the path thickening step is the larger part of the total execution time. It can be as high as hundreds of milliseconds. However, doubling the baud rate would reduce this time by 50% allowing for adaptation of the algorithm performance to the application requirements. Path length is affected by the distribution of unavailable nodes in the physical grid and can increase by as much as 106%. This distribution also increases the execution time of the initial path generation process but decreases it for path thickening.

Since the path thickening step is the longer to execute, future work will focus on reducing it by developing more efficient neighbor interrogation mechanisms. Adaptations for wireless nodes will also be explored.

## References

[1] E. Lee, "Cyber physical systems: Design challenges," in *University of California, Berkeley Technical Report No. UCB/EECS-2008-8*, 2008.

[2] K. Passino, *Biomimicry for Optimization, Control, and Automation*. Springer, 2005.

[3] F. Zhao, C. Bailey-Kellog, and M. Fromherz, "Physics-based encapsulation in embedded software for distributed sensing and control applications," *Proceedings of IEEE*, vol. 91, no. 1, pp. 40–63, January 2003.

[4] C.-T. Chen, *Linear System Theory and Design*. Oxford, 1999.

[5] J. Han and M. Kamber, *Data Mining. Concepts and Techniques*. Morgan Kaufman, 2006.

[6] T. He, B. Blum, J. Stankovic, and T. Abdelzaher, "Aida: Adaptive application-independent data aggregation in wireless sensor networks," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 2, pp. 426–457, May 2004.

[7] W. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive protocols for information dissemination in wireless sensor networks," in *Proceedings of MobiCOM*, 1999, pp. 174–185.

[8] C. Intanagonwiwat, D. Estrin, H. Govindan, and J. Heidemann, "Impact of network density on data aggregation in wireless sensor networks," in *Proceedings of International Conference on Distributed Computing Systems*, 2002.

[9] C. Lu, B. Blum, T. Abdelzaher, J. Stankovic, and T. He, "Rap: A real-time communication architecture for large-scale wireless sensor networks," in *Proceedings of IEEE RTAS*, 2002.

[10] P. Wang, C. Li, and J. Zheng, "Distributed data aggregation using clustered slepian-wolf coding in wireless sensor networks," in *Proceedings of International Conference on Communications*, 2007, pp. 3616–3622.

[11] T. Banerjee, K. Chowdhury, and D. Agrawal, "Distributed data aggregation in sensor networks by regression based compression," in *Proceedings of MASS*, 2005.

[12] S. Madden, M. Hellerstein, and W. Hong, "Tag: A tiny aggregation service for ad-hoc sensor networks," in *Proceedings of ACM Symposium on Operating System Design and Implementation*, 2002.

[13] V. Subramanian and A. Doboli, "Online adaptation policy design for grid sensor networks with reconfigurable nodes," in *Proceedings of Design, Automation and Test in Europe Conference*, 2009.

[14] "Cypress semiconductor corporation, psoc mixed signal array," *Document No. PSoC TRM 1.21*, 2005.