Communication Network Control

Deadlock Free Routing Algorithm – Graphic Simulation

Author: Adi Lavi
027448166
Technion, Spring Semester 2009

1 Introduction:

This document concludes the final project in course "Communication Networks Control". It includes design and implementation overview and user manual for a graphic simulation of Efficient Deadlock-Free Routing, by B. Awerbuch, S. Kutten and D. Peleg [1]

The simulation provides continuous "real-time" demonstration of the algorithm operation and can be used for training purposes

2 Notes on the Article

1. It is stated that when a token travels from vertex v to vertex w, 1 of the test conditions that has to be fulfilled for the traverse to take place, is that it has to be the turn of v in w incoming edges. This is a redundant test, because if a buffer in w is empty, it is enough for the token to traverse.

   To emphasize this point, let's look at another point in the algorithm for vertex w: Each vertex w (according to the algorithm) maintains an InQueue (at level i) of neighbors with tokens that are currently awaiting entrance to w on level I (all tokens that got stuck because w was not empty when they tried to traverse to w). As soon as a buffer in w becomes empty, a background "releasing-task" releases the stuck token that is at the top of the queue waiting for this buffer.

   There is no need to maintain an additional queue to the InQueue. Assuming the buffers in queue must be empty to allow the traverse, we look at 2 possible cases that affect the interaction with the queue: first, the buffers in w are full and one of them becomes empty – then, if the queue is not empty, then immediately the top waiting token is put into the emptied buffer and removed from the queue. The buffers are full again. Second, at least one the buffers are empty. That implies that the InQueue is empty (otherwise, it would contradict case #1), therefore it is safe for the token to traverse from v to w.

2. When a "merge" operation occurs, the token that gets locked is necessarily waiting at the InQueue of the next vertex (the condition for a "merge" is that the buffers at w are full, and there is another token in v that was stuck). The waiting token **must** be removed from the waiting list at w. The reason is that this token would never be released from the InQueue at level i (the level it got stuck in), because it will progress at a higher level (the next step after it gets locked is for it to be freed by a Check process, and upgraded 1 level up). Moreover, if it remains at the InQueue of w at level i, since it cannot be processed into w at level i (because it's locked), it would remain as "zombie" and block other tokens in the queue to be released into w at level i once a buffer is emptied.

3. It is necessary to state that when a Check process checks every edge (u,v) such with a positive Debt and the Int_Debt(i) that is positive, the order of the Debt

counters must place the Int_Debt(i) as the last in the Debt list (see pseudo-code of Check Process in page 185 of the article). The reason for that is that if the Int_Debt(i) counter is not last, once the Check process reach it, the Check practically finishes its work at level i either by releasing token t' (that got locked by the "merge" operation), or by releasing the locked buffers and spawning 2 Check processes at level i+1. In both cases, the algorithm states that this Check process terminates. This is wrong because by doing that, if the Int_Debt(i) is not last in the debt list, the Check process may lose counters at level i it should have released, therefore we might end up with unreleased tokens / buffers, thus a deadlock.

3 Installation

3.1 Open the 'deadlock-free-routing-ui.zip' file using Winzip or Winrar. Extract all its contents into a selected location.

3.2 Installing Java Runtime Environment:

Double-click '**jre-6u16-windows-i586.exe**' and follow its setup instructions.

4 Starting the Application:

Right-click the executable file “deadlock-free-routing-ui.jar”. Select “Open With” -> “Java™ Platform SE Binary Runtime”

5 Basic Assumptions

5.1 The graph is finite and directed
5.2 Each node has a unique identity
5.3 Weights of edges are irrelevant
5.4 Calculation done by nodes is negligible
5.5 All messages are delivered in a finite time
5.6 Messages travel from node I to node j in maximum 1 unit of time
5.7 Message size is negligible
5.8 The program uses Bellman-Ford shortest-path algorithm to compute that path from node j to node j

6 Implementation Overview

This program is composed of 2 parts: core and user interface. It is written in Java and uses Java Swing for the user interface part.

6.1 Application Core

The application core is the part that receives a directed graph as input, where the graph simulates a network, and executes the algorithm on this graph. The purpose of the core is to follow the algorithm's steps and verify that all participating tokens reach their final destination, and a deadlock is never created in the graph.

6.1.2 Token traverse

Each token has a pre-registered source vertex and destination, where the path is selected by using Bellman-Ford shortest path algorithm, and is stored as a pre-execution parameter for each token.

6.1.3 Core operations

The operations the are performed during the execution are as follows: introduction of a token in a vertex, traverse of a token between 2 vertices, entry of a token into a waiting list of a vertex (when the vertex is full), release of a token from waiting list into the vertex, lock of a token and a vertex, release of a locked token, and release of a locked vertex.

Each operation of the algorithm is synchronous, no other operation is allowed during a traverse of a token over an edge, or a merge, or a release operation.

6.2 User Interface

The user interface displays the graph and the algorithm execution. It enables 2 major functions:

1. Creation and manipulation of graphs. A graph can be manually created. In addition, a graph can saved as a .properties file to the File System and also can be loaded from a .properties file.
2. Running the algorithm. During the execution of the algorithm, the user can zoom into every step to understand its core. Zooming into the objects is also available, in order to view the state of each vertex and token during each step of execution.

7 Using the Application:

The application GUI is composed from 2 main panels:

1. Top Panel – control panel for graph manipulation and for running and tracing the algorithm.
2. Bottom Panel – displays the graph

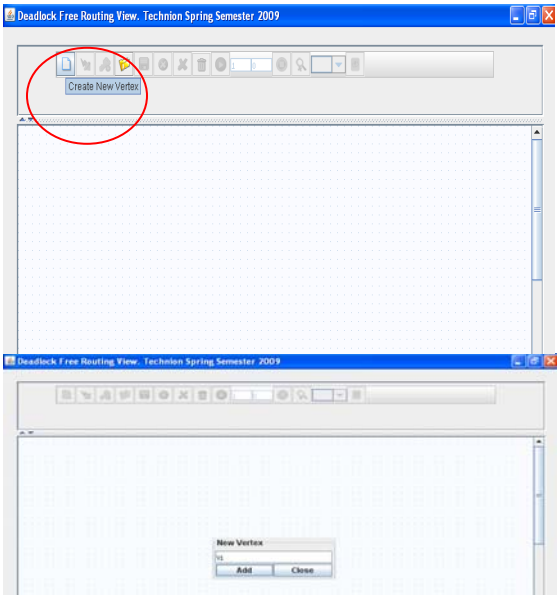
7.1 Graph Creation

It is possible to create a new graph from scratch, using the control panel.
Simply start by adding new vertices, and connect them with edges.
Once the graph is created, define tokens to traverse in the graph, each has a source and a target vertex. Tokens represent messages in a network.
It is also possible to delete existing vertices, edges and tokens.

7.1.1 Add New Vertex

Click on the “Create New Vertex” icon. A popup window will be opened. You need to define the vertex’s name, and then click on “Add”. You will see the new vertex added to the graph. The window will remain open to enable adding another vertex immediately.
In order to close the window, click on “Close”. If a vertex name field was filled, it will be discarded.
Vertex name must be unique (case-insensitive) and cannot be empty. The maximum length for a vertex name is 10 characters.

Figure 7-1-1: Add New Vertex



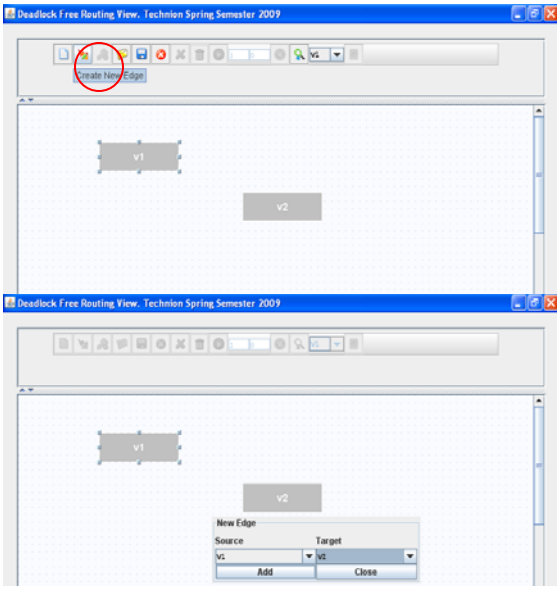
7.1.2 Add New Edge

Click on the “Create New Edge” icon. A popup window will be opened. You need to select the edge’s source and target vertices, and then click on “Add”. You will see the new edge added to the graph. The window will remain open to enable adding another edge immediately.

In order to close the window, click on “Close”. If a source or target vertex were selected, it will be discarded.

An edge cannot connect a vertex to itself.

Figure 7-1-2: Add New Edge



7.1.2 Add New Token

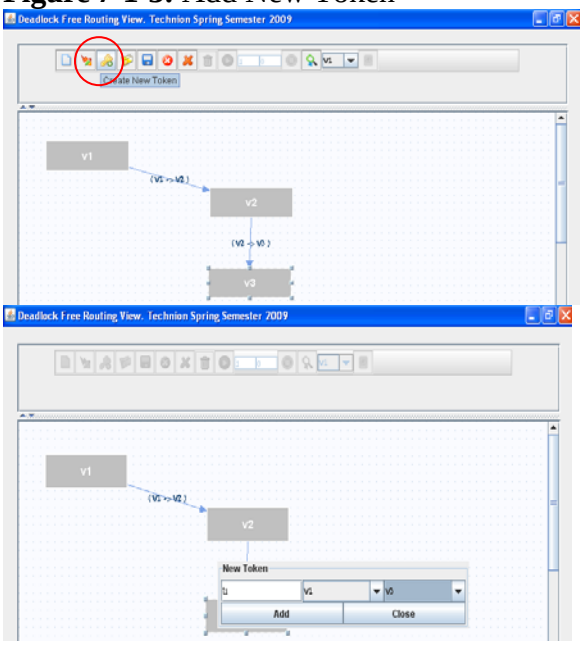
Click on the “Create New Token” icon. A popup window will be opened. You need to select the start vertex target vertex of the token, and then click on “Add”. You will see the new token added on the control panel. When you hover over the token you will see the path that was assigned to the token as its tool tip. The application uses Bellman-Ford’s shortest path algorithm to calculate token paths. The token is grayed out until it is introduced to the network.

The window will remain open to enable adding another token immediately. Its source vertex will display it as it waits to be introduced in this vertex.

In order to close the window, click on “Close”. If a source or target vertex were selected, they will be discarded.

Token name must be unique (case-insensitive) and cannot be empty. The maximum length for a token name is 10 characters.

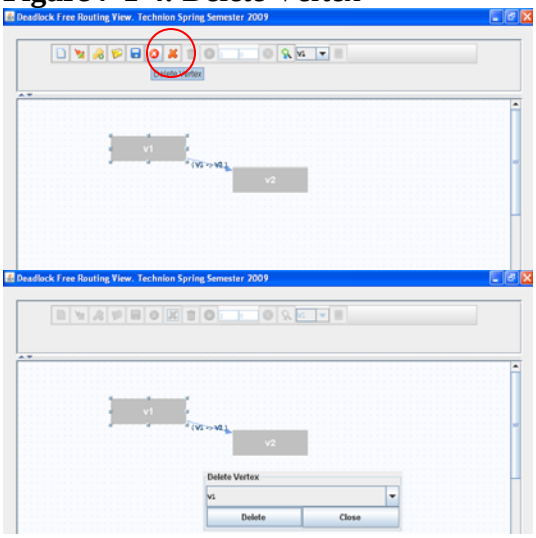
Figure 7-1-3: Add New Token



7.1.4 Delete Vertex

Click on the “Delete Vertex” icon. A popup window will be opened. You need to select the edge to be deleted, and then click on “Delete”. In addition to the deleted vertex, all incoming and outgoing edges from this vertex are deleted. In addition, tokens with a path that included this vertex get a new path. If a path does not exist for a token, the token is deleted. The window will remain open to enable deleting another vertex immediately.
In order to close the window, click on “Close”. If a vertex was selected, it will be discarded.

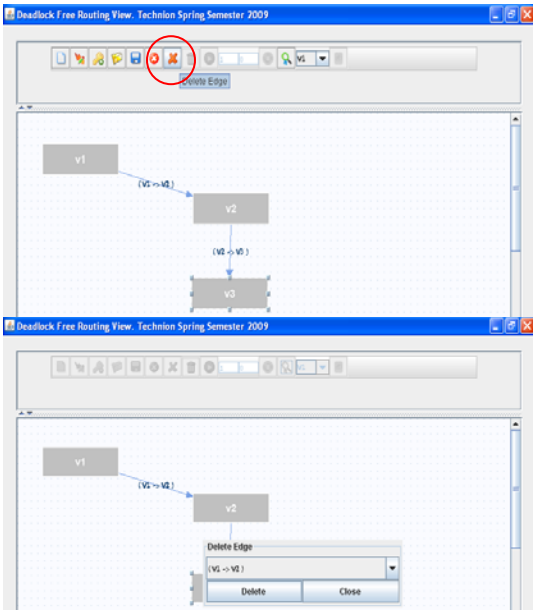
Figure 7-1-4: Delete Vertex



7.1.5 Delete Edge

Click on the “Delete Edge” icon. A popup window will be opened. You need to select the vertex to be deleted, and then click on “Delete”. In addition to the deleted edge, tokens with a path that included this edge get a new path. If a path does not exist for a token, the token is deleted. The window will remain open to enable deleting another edge immediately.
In order to close the window, click on “Close”. If an edge was selected, it will be discarded.

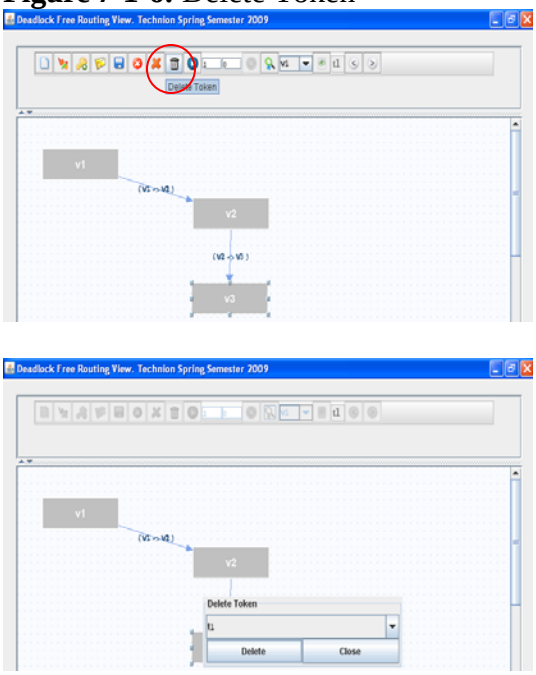
Figure 7-1-5: Delete Edge



7.1.6 Delete Token

Click on the “Delete Token” icon. A popup window will be opened. You need to select the token to be deleted, and then click on “Delete”. The window will remain open to enable deleting another token immediately.
In order to close the window, click on “Close”. If a token was selected, it will be discarded.

Figure 7-1-6: Delete Token



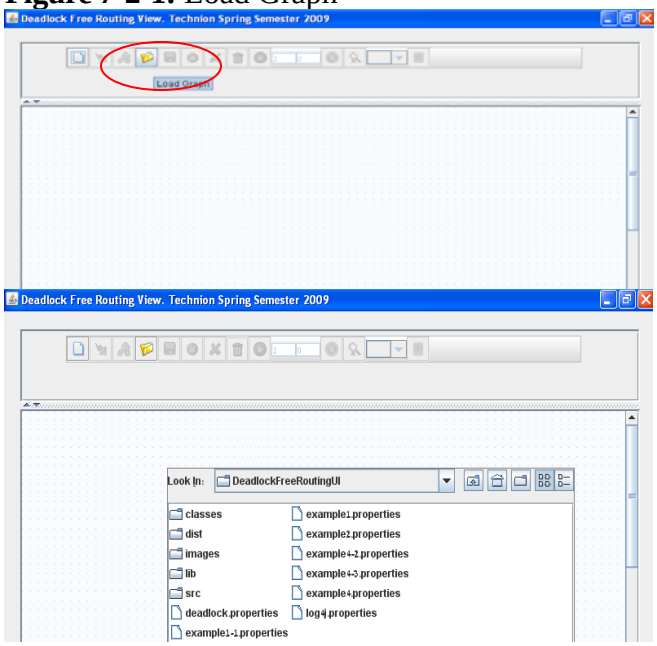
7.2 Loading and saving graphs

It is possible to load a graph that was previously saved to a file, from the file system. It is also possible to save the current graph to the file system. The graph configuration is stored in a .properties file.

7.2.1 Load Graph

Click on the “Load Graph” icon. A popup window will be opened. You need to select a “.properties” file, and then click on “Open”. The selected graph configuration will be loaded, overriding any existing graph on the graph panel that was previously handled. If you click on the “Cancel” button, the window will be closed without loading any graph, leaving the current state as it was.

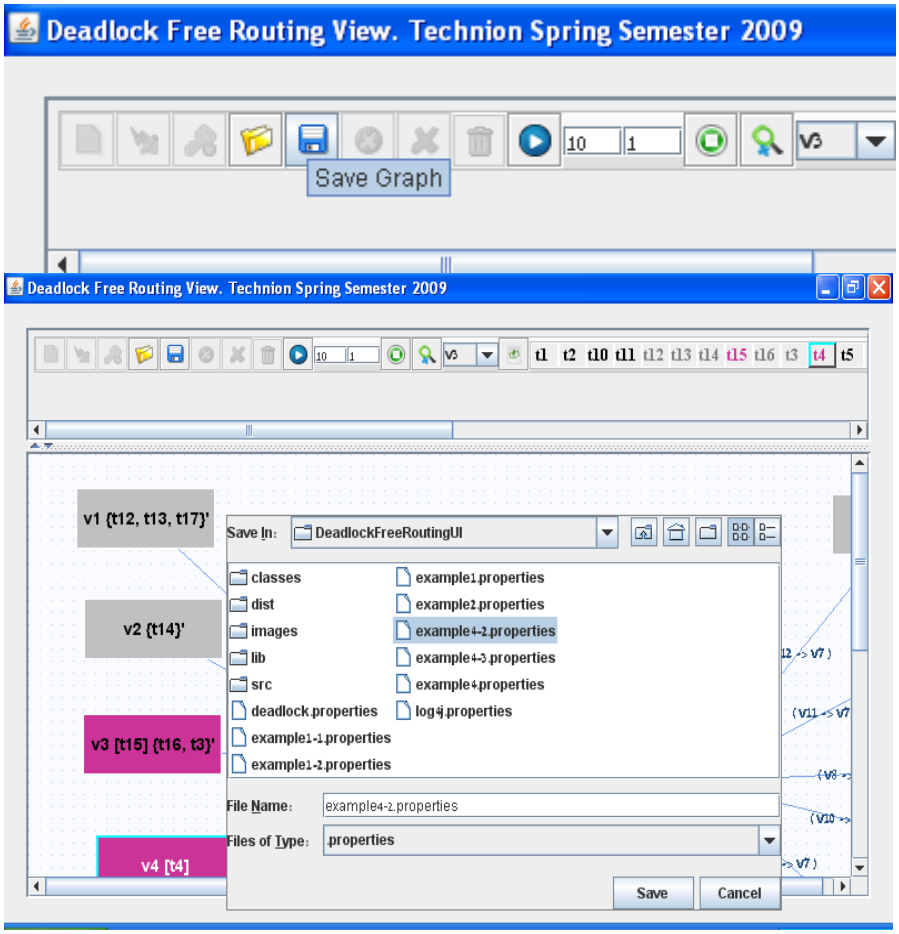
Figure 7-2-1: Load Graph



7.2.2 Save Graph

Click on the “Save Graph” icon. A popup window will be opened. You need to select a “.properties” file, and then click on “Save”. The graph’s configuration will be stored in the file, storing all vertices, edges and tokens. If you select an existing file, it will be overridden.
If you click on the “Cancel” button, the window will be closed without saving.

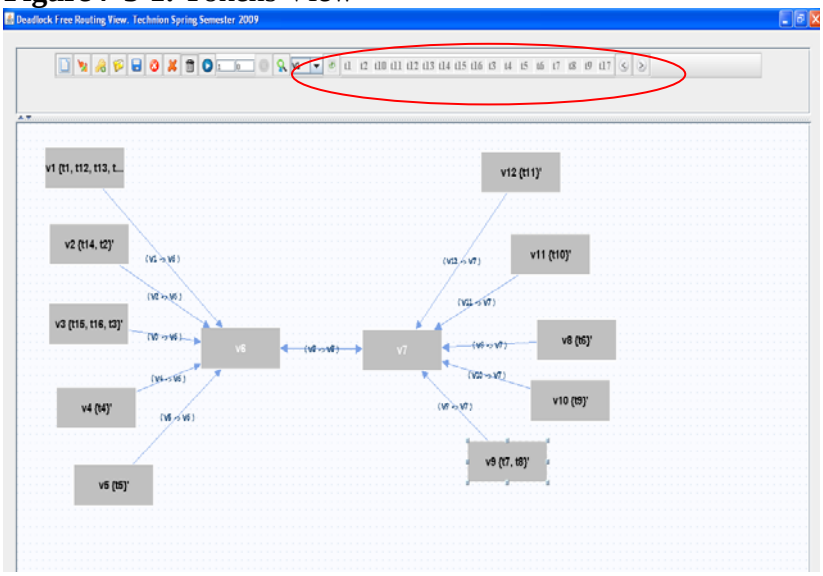
Figure 7-2-2: Save Graph



7.3 Tokens View

The application applies the algorithm to traverse all defined tokens through the path that was calculated for each token. Before running it, you can observe the tokens to be processed and affect the order of processing.
The application makes its steps by moving on all the tokens in a round-robin queue, where each token takes its next step when its turn arrives in the queue. The queue is ordered by default alphabetically, and is displayed by the control panel. It is possible to change the tokens order by moving them to the right or to the left before running the algorithm.

Figure 7-3-1: Tokens View



7.4 Running the Algorithm

In order to run the algorithm, click on the “Run” icon. The algorithm would process all defined tokens to their destinations. A successful run means that all launched tokens have reached their destinations (i.e. no deadlocks were created).

As previously mentioned, the application processes all tokens in a round-robin queue, where the next step is performed either on the next token in the queue, or for a Check process that was triggered by a token that reached its destination.

7.4.1 Controlling the steps

The user can define the following before clicking the “Run” icon:

1. How many steps the application will take from its current state (default is 1 step). If the defined steps number is empty or non-positive number, it hints the application to run the algorithm until its end). Otherwise, it will stop after the define number of steps.
2. Sleep time in seconds between each step (default is 0, i.e. no wait). If the defined number of steps was not 1, then the application will wait X seconds (as defined) between steps. This way the user can observe the ongoing changes during the run in “slow motion”.

7.4.2 Run

Click on the “Run” icon to run the algorithm. The application will obey the configuration (see section 5.3.2.1) and run X steps with Y sleep time after every step.

7.4.3 Reset

At any point of the algorithm, is it possible to reset the state of the graph to its initial state and run it from the start. All vertices and tokens will be reset. Click on the “Reset” icon to reset the graph.

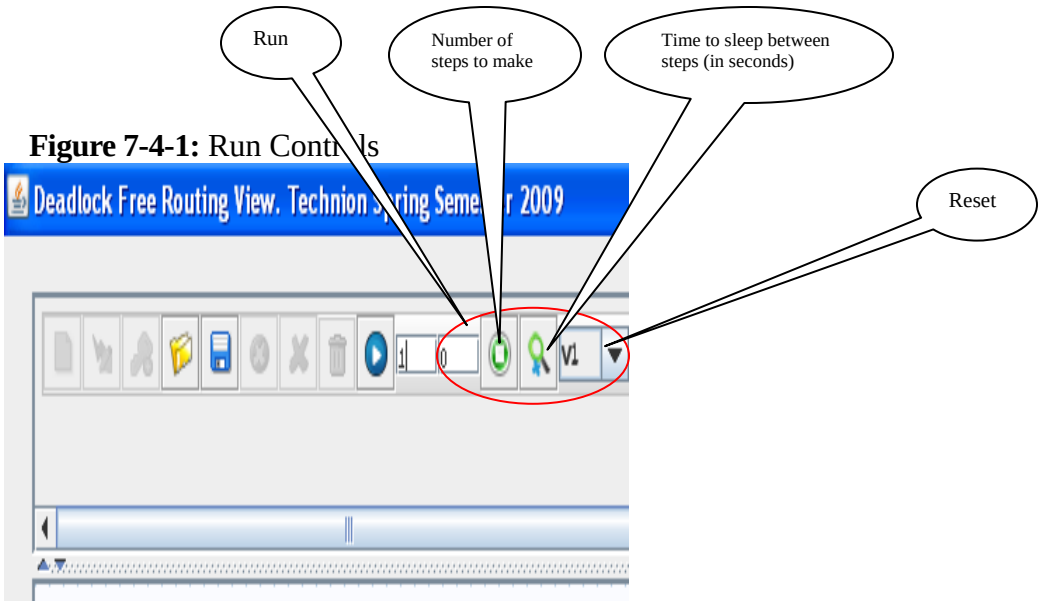


Figure 7-4-1: Run Controls

7.4.4 Vertices State

The graph displays the state of each vertex as follows:

1. Vertex state:
    - Gray – the vertex is empty, i.e. all layers are empty
    - Orange – the vertex is active, i.e. none of the layers is stuck or locked, and at least 1 token is in this vertex
    - Purple – the vertex is stuck, i.e. at least in one of its layers there is a stuck buffer (that contains a stuck token)
    - Red – the vertex is locked, i.e. at least in one of its layers there is locked (when a vertex contains both a stuck and a locked buffer, the displayed state is "locked")
    - Green border – when a vertex is the destination of a token and the token arrived at its destination, the borders of the vertex are marked in green
2. Vertex's tokens display:
    - Starting tokens – if there is at least 1 token that is waiting to be introduced in a vertex, the contents of the vertex is marked in black font. The list of tokens waiting to be introduced is marked with '. If there are no tokens waiting to be introduced, the contents of the vertex is marked in white font
    - Active tokens – all tokens that reside in a vertex are displayed in the vertex (each layer surrounds its tokens with a [parenthesis)
    - Waiting tokens – stuck tokens are displayed both in the vertex they are stuck in, and in the vertex they wait to enter in its waiting list. The list of waiting tokens is marked with '!'

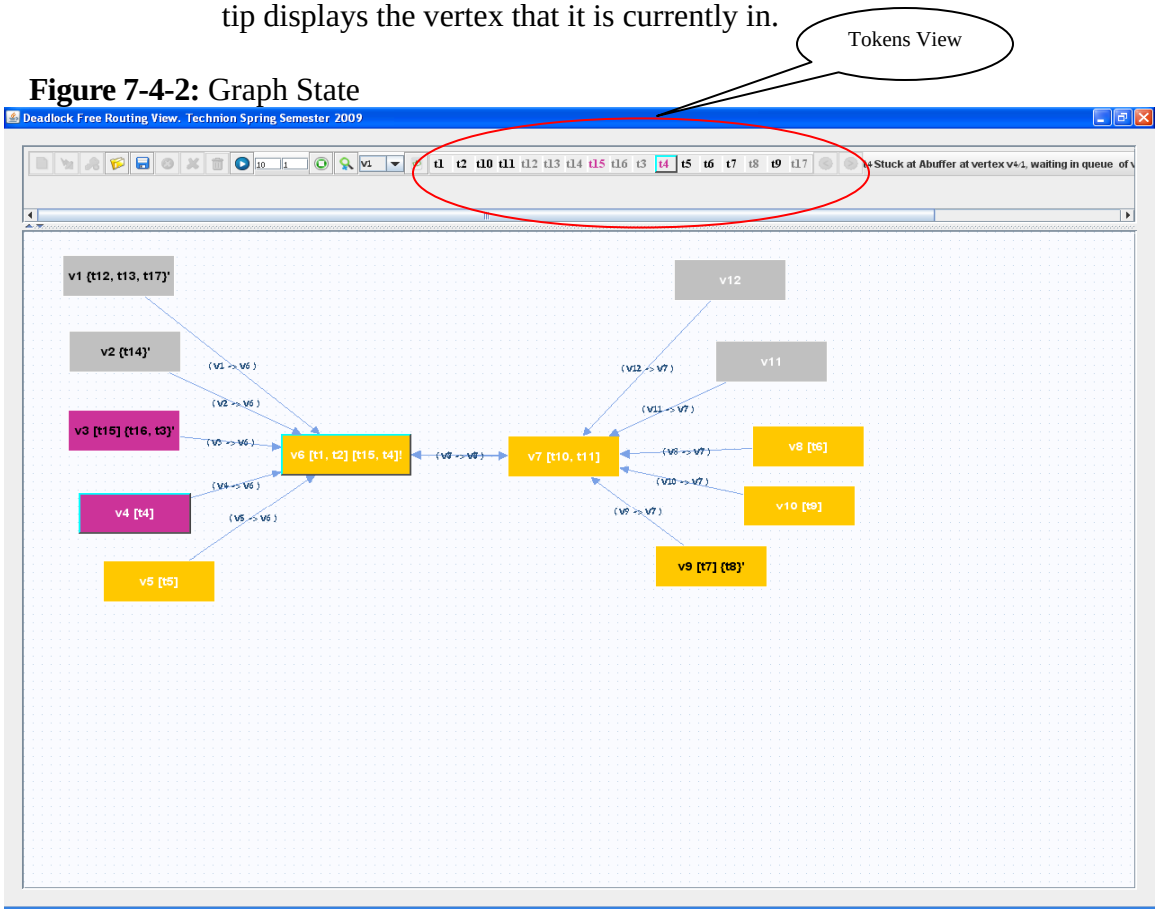
- Locked tokens – if a vertex is locked, its locked tokens are marked with '*'

7.4.5 Tokens state:

The control panel displays the state of each token as follows:

- Gray – the token was not introduced to the network
- Black – the token is active
- Purple – the token is stuck
- Red – the token is locked
- Green – the token reached its destination and terminated
- During run, for each token, in addition to its path, its tool tip displays the vertex that it is currently in.

Figure 7-4-2: Graph State



7.4.6 Tracing the algorithm

There are several ways to trace the inside of the algorithm while it is in run. After each step, the application does the following:

1. Updates and displays the state of all vertices and tokens (see section 5.3.2.3)
2. Marks the border of the vertices and tokens that participated in the last step in light blue
3. Event message area:

The event message area displays the last step that was performed, and gets updated after each step

4. Zoom-In a vertex:
   It is possible to zoom into a vertex and view its internals, i.e. the state of each layer and each buffer in the vertex at. To zoom-in, click the “Zoom-In” icon in the control panel. A window will be opened to zoom into the vertex that was selected. The semantics of the colors in the zoom-in view is the same as in the graph.
   It is possible to leave the zoom-in window open, proceed with the run, and click the “refresh” button to refresh the view of the vertex.
5. Trace token view:
   It is possible to trace a particular token. Click on the “Trace token” icon. A window will be opened, where you need to select the token and click “Show”. All events that included the selected token will be displayed in the order of

Figure 7-4-3: Zoom-In to vertex

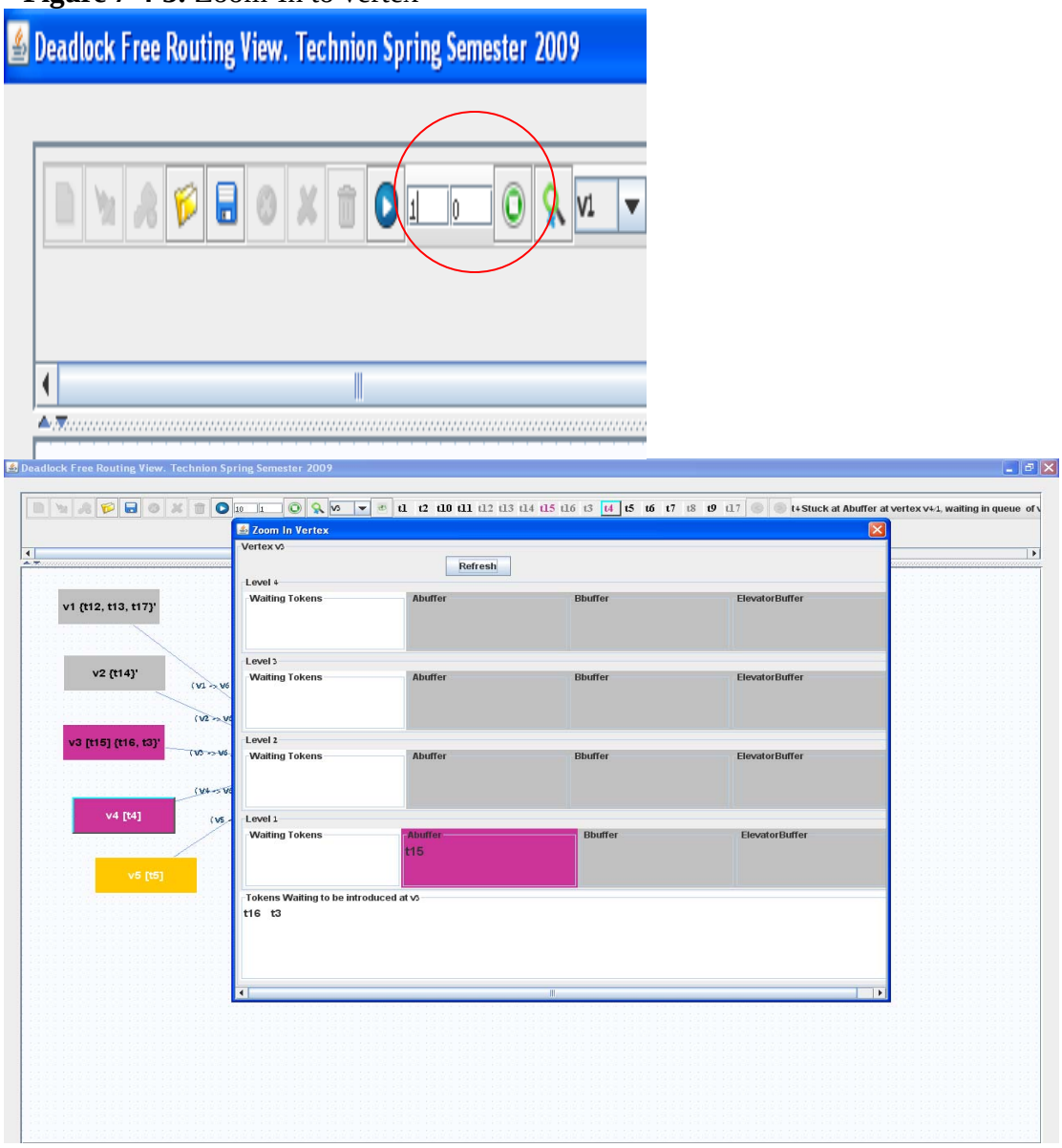


Figure 7-4-4: Trace Token

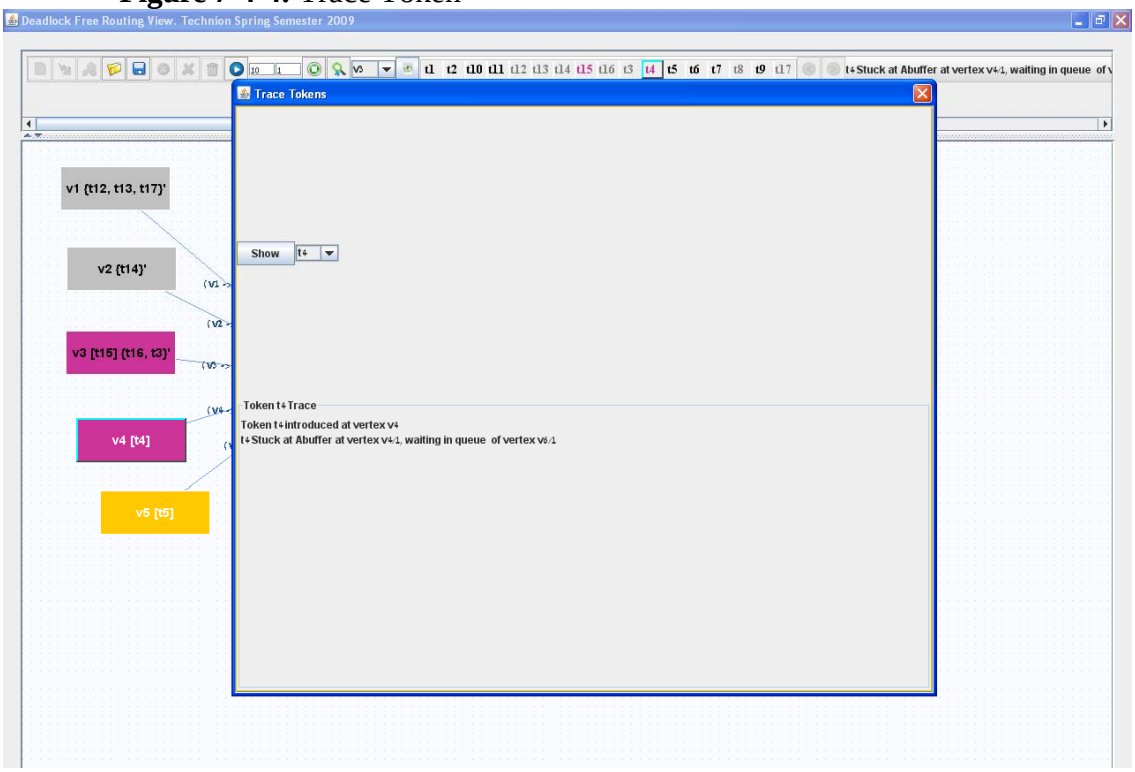
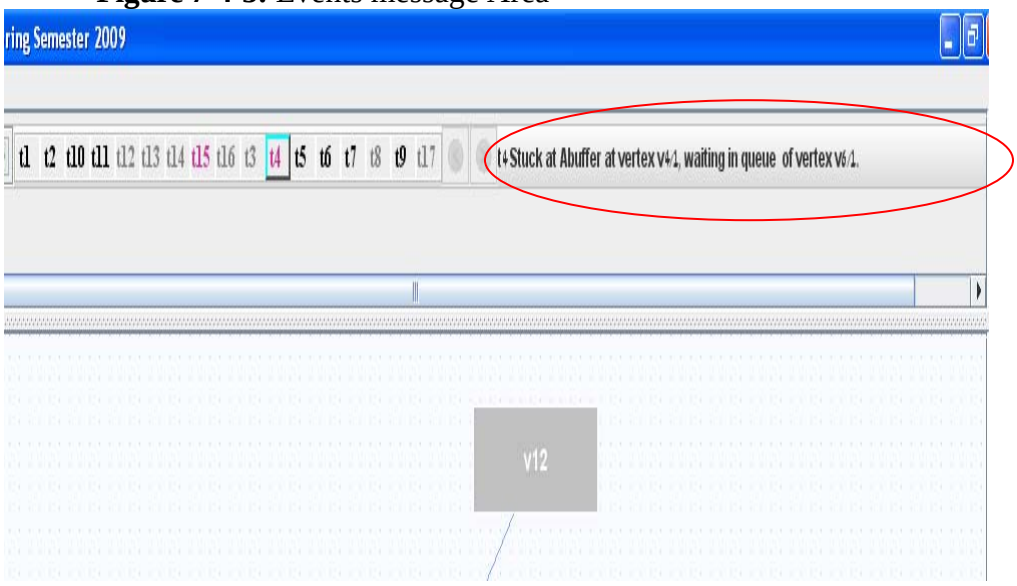


Figure 7-4-5: Events message Area



References:

[1] B. Awerbuch, S. Kutten and D. Peleg: "Efficient Deadlock-Free Routing", Proceedings of the 10th ACM Annual Symposium of Principles of Distributed Computing (PODC 91), Montreal, Quebec, Canada, August 1991