

# Automating the Presentation of Computer Networks

**G. Vandenberghe and J. Treurniet**

Defence R & D Canada – Ottawa  
3701 Carling Avenue  
Ottawa, Ontario K1A 0Z4  
CANADA

[Grant.Vandenberghe@drdc-rddc.gc.ca](mailto:Grant.Vandenberghe@drdc-rddc.gc.ca); [Joanne.Treurniet@drdc-rddc.gc.ca](mailto:Joanne.Treurniet@drdc-rddc.gc.ca)

## 1. ABSTRACT

*There are several graph layout algorithms available to automatically display a computer network. This study applies seven existing layout algorithms to a computer network and compares their readability, complexity, and speed. These algorithms generate network diagrams that are difficult to quickly interpret. To address this issue two improved layout algorithms called the XY Control algorithm and Voting algorithm were developed. The XY algorithm is a new type of force-directed model with an improved grid-like appearance relative to other force-directed algorithms. The Voting algorithm is a directed hierarchal layout technique that provides better space utilization while minimizing edge crossings. Both new algorithms are comparable in speed to the existing algorithms.*

## 2. INTRODUCTION

System administrators use network diagrams for documenting, planning, and controlling computer networks. Initially, generic drawing packages like Microsoft PowerPoint [1] were used to do network drawings. Since then other tools like Visio [2] that better maintain the relationships between nodes and interconnecting links have been developed. The continuous growth of computer networks has created the need for more specialized documenting and planning tools such as Smart Draw [3], NetViz [4] and OPNet Modeler [11]. All of these manual-drawing packages are quite powerful and feature-rich but offer a static view of the network with negligible automatic layout features.

Network management tools enhance basic network viewing by overlaying dynamic state information. Packages like Nimsoft [5] have integrated the ability to read Visio's file format with software to overlay operational state information. Other network management tools like Computer Associates Unicenter [6,7] generate internal network drawings using multiple windows that are overlaid on top of a summary screen. HP OpenView [8,9] is able to automatically arrange nodes using either a force-directed approach or hierarchal approach. Most network management systems include both automated layout and manual placement options making it difficult to judge from a simple literature search the quality of the layout algorithms used. Demo versions of several network management products were reviewed including LAN Surveyor [12,13], NetCrunch [14,16], What's Up Gold [15], and OpManager [17]. In summary, NetCrunch and OpManager had the strongest graphical network presentation capabilities of those reviewed. However the automatic layout capabilities of all these tools were limited and required manual placement assistance.

A number of software libraries [20] offer a wealth of automatic layout algorithms and presentation features. Function libraries like JGraph [18] and Tom Sawyer Layout [19] include several established layout algorithms. The layout algorithms are generic but not always well suited to the display of computer networks, with some being loop intolerant and others presenting computer network information in a non-intuitive manner.

Vandenberghe, G.; Treurniet, J. (2006) Automating the Presentation of Computer Networks. In *Visualising Network Information* (pp. 1-1 – 1-18). Meeting Proceedings RTO-MP-IST-063, Paper 1. Neuilly-sur-Seine, France: RTO. Available from: <http://www.rto.nato.int/abstracts.asp>.

This study compares nine automatic layout algorithms in terms of their strengths and weaknesses as they relate to an enterprise computer network. Seven of the algorithms are based on established techniques while the remaining two are newly developed algorithms named the XY control and Voting algorithms. These new layout algorithms are expressly tuned to the needs of a computer network giving them an advantage over the generic layout approaches.

Benchmarking criteria that included planarity [minimal edge crossings], speed, and code complexity were established to compare the various layout algorithms. A benchmark network, shown in Figure 1, was used as part of the comparison study. The network includes a subnet with redundant computer connections, an optical ring, and several non-redundant clusters.

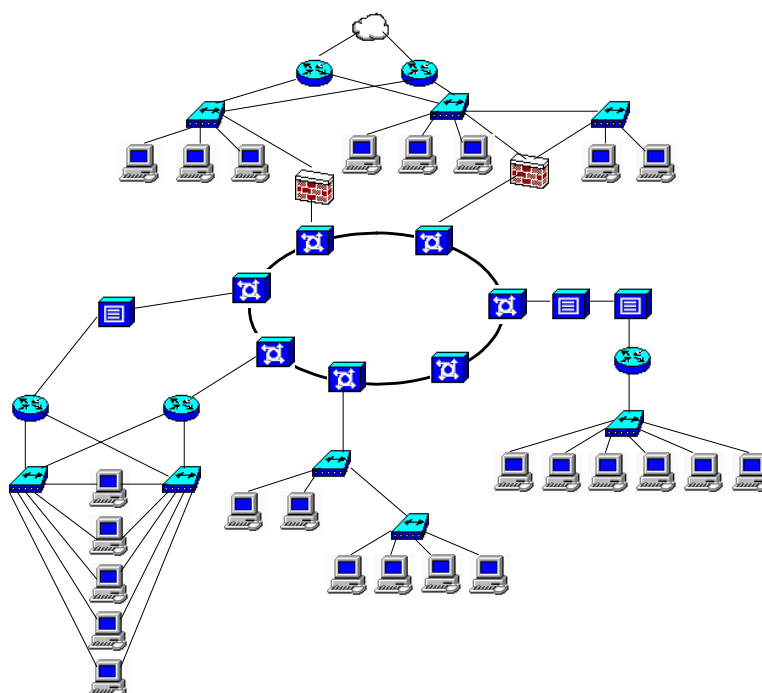


Figure 1. The benchmark network, manually positioned.


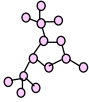
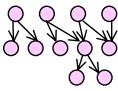
### 3. EXISTING LAYOUT ALGORITHMS

There are a large number of layout algorithms available in the open literature; a survey of techniques is available in [31] [32]. The seven most applicable to computer networks are listed and referenced in Table 1 along with their graphical classification. The three graphical classifications are described below:

- **Tree Graph:** A tree graph is a loop intolerant graph with a single root vertex. Many computer networks can be drawn using tree-like principles once the effects of the loops are ignored through a spanning tree algorithm [21] and hidden vertices are added to create the illusion of a single root vertex.

- **Undirected Graph:** Undirected graph algorithms try to keep associated devices together and unrelated devices spread apart. Algorithms reviewed in this classification generate the network diagrams using a force or energy directed algorithm<sup>1</sup>. The free form of an undirected graph makes them a popular choice for representing computer networks. HP OpenView [8] and Intellitactics NSM [10] have both used undirected graph techniques to represent a computer network.
- **Directed Graph:** Directed graph algorithms include a strongly embedded concept of hierarchy that forces nodes of a similar level to share a common row of the graph. Computer networks can be converted into a directed graph by imposing a hierarchy on the links and can be quite natural-looking once the horizontal placement of nodes on a level is allowed to vary.

Table 1. Selected Layout Algorithms

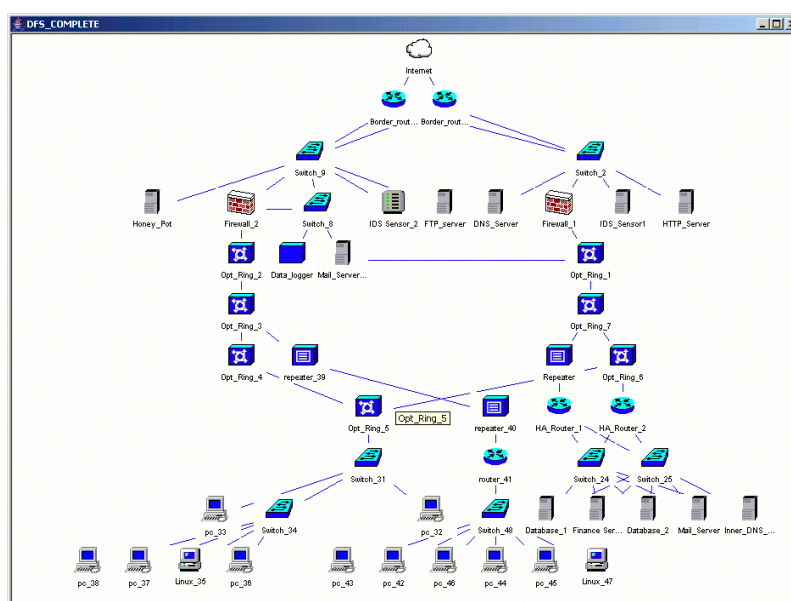
GRAPH CLASSIFICATION	LAYOUT ALGORITHM NAME
Tree Graph 	Depth First Search [21]
Undirected Graph 	Spring Embedder [22][23]
	Magnetic Spring* [23]
Directed Graph 	Barycenter [23][24]
	Median [28]
	DFS with Barycenter [25][33]
	Sifting Algorithm [29][30]

\* The Magnetic Spring approach includes hierarchal concepts usually associated with directed graphs. However, it is listed as an undirected approach because it is an extension of the Spring Embedder approach.

### 3.1 Depth First Search

Computer network diagrams that are drawn by hand frequently bear some tree-like resemblance. If a network is non-redundant (contains no loops) then a layout technique such as the Depth First Search (DFS) is appropriate because of its speed and simplicity. Unfortunately, DFS is loop intolerant and a spanning tree algorithm is needed to break the loops. Results from the DFS with spanning tree are shown in Figure 2. There are 14 avoidable edge crossings. Although the algorithm does not deal with edge crossing, the overall graph planarity is surprisingly similar to many of the directed graph approaches that are examined later.

<sup>1</sup> Using a directed algorithm on a undirected graph seems contradictory, however an undirected graph reflects the nodes relationship with its link, while a force-directed algorithm reflects a force application on the nodes.



**Figure 2. The Depth First Search With Spanning Tree algorithm applied to the benchmark network.**

## 3.2 Spring Embedder

The Spring Embedder approach tries to balance a set of conflicting attractive and repulsive forces to arrange the nodes on a network diagram. A typical result for the benchmark network is shown in Figure 3. The diagram bears little resemblance to the original diagram (Figure 1). The redundant links cause the algorithm to get locked into a local minimum which results in 20 avoidable edge crossings. This investigation found that the results from the algorithm are inconsistent because it starts from a random initial sequence and is non-hierarchical.

## 3.3 Magnetic Spring

The Magnetic Spring approach draws upon the capabilities of the Spring Embedder algorithm but adds a compass-like rotation to the interconnecting links. Using a user-defined network hierarchy as a base, the magnetic spring approach tries to orient all the links in the same direction. This approach is significantly better than the Spring Embedder approach as can be seen in Figure 4. The network diagram is more readable, however the technique still has 16 avoidable edge crossings and the nodes are cluttered in some parts of the graphical presentation.

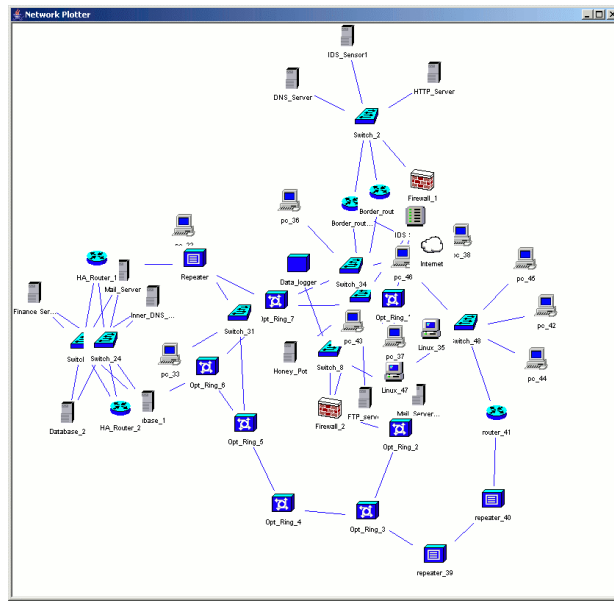


Figure 3. The Spring Embedder algorithm applied to the benchmark network.

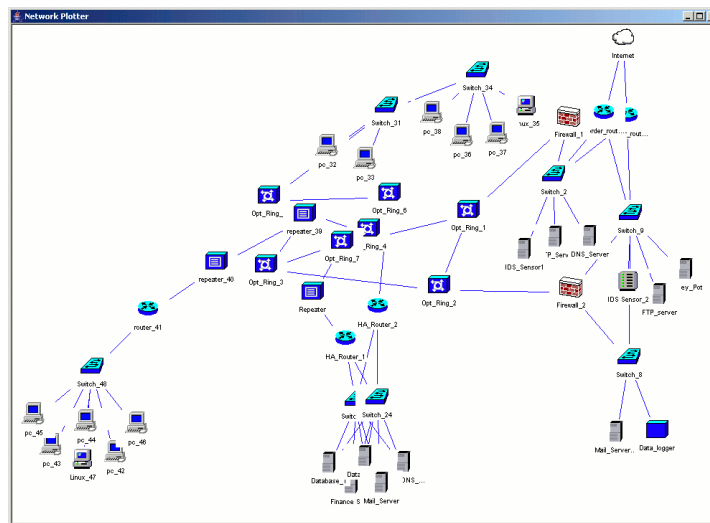


Figure 4. The Magnetic Spring algorithm applied to the benchmark network.

### 3.4 Barycenter, DFS with BaryCenter, Median, and Sifting Algorithms

The Barycenter, Median, and Sifting algorithms arrange the computer network into rows and then proceed to alter the sequencing within the rows to minimize edge crossings. Each technique uses a slightly different criteria for sequencing the nodes but the general approach is similar. All of the algorithms use an up-down repeated sweeping action and an edge-crossing algorithm to measure improvement. The sifting algorithm tends to generate the best results but takes longer to complete. The Median approach provided

the best compromise between quality and speed. Since the appearance produced by the Barycenter, Median and Sifting algorithms is similar, only the view generated by the Median layout algorithm is provided in Figure 5. These approaches do a reasonable job of arranging the network but they all tend to get locked into a local minimum because of fork-like structures in the benchmark network. The total number of edge crossings for the Barycenter, Median and Sifting algorithms are 18, 14, and 13 respectively.

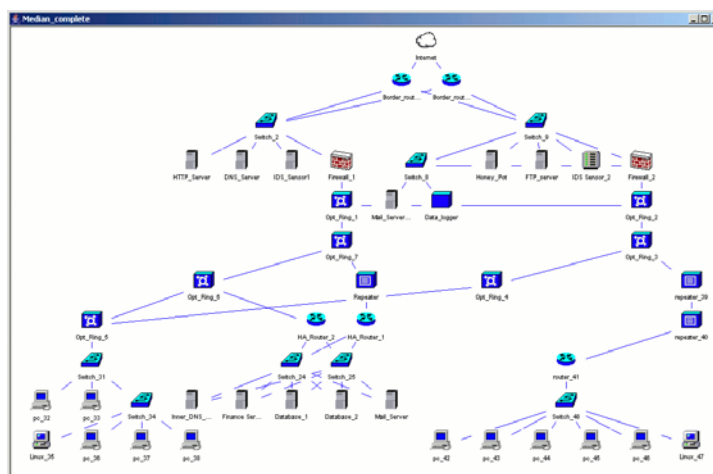


Figure 5. The Median Layout algorithm applied to the benchmark network.

In an attempt to improve these results some effort was made to organize the random placement of nodes entering the algorithm. The literature has shown that prefacing the Barycenter approach with a Depth First Search improves graph planarity and speed [36,37]. This is true for a densely crossed generic network, however because of the fork-like structures present in the sparsely-crossed benchmark network the algorithm tended to get locked into local optima much faster when this approach was taken. In this case, the Barycenter with DFS approach tended to look very much like the DFS approach and had a similar number of edge crossings for the benchmark network (14).

## 4. CUSTOM LAYOUT APPROACHES

### 4.1 XY Control Algorithm

The XY Control algorithm is a new custom layout technique that is based on industrial control system. The algorithm uses a set of conflicting forces similar to the Spring Embedder and Magnetic Spring approaches but is more complex. The XY algorithm uses five different control systems that are split along the X and Y-axes:

1. **Hierarchal Controller:** The hierarchal controller applies a force to any device that falls outside of its preferred horizontal band. This controller is essential for establishing the hierarchy in the diagram.
2. **Astable Controller:** The Astable non-linear controller measures the vertical distance between nodes and tries to control their vertical proximity. The term astable is used because there are multiple points in space where each node can come to rest. As shown in Figure 6, the non-linear proportional controller (the box with the triangular function in it) essentially allows the mathematical model for a standard second order system to come to rest at more than one point in

space. The graph shown beneath the control system diagram in Figure 6 shows a stepped response for two different starting positions. As can be seen, each end result is stable but different. This difference allows the XY Control algorithm to arrange devices in two rows which helps reduce the graph's width.

3. **Attractive Controller:** The attractive controller operates in the horizontal direction and is intended to align related devices in a vertical column.
4. **Repulsive X Controller:** This repulsive controller operates in the horizontal direction. The repulsive controller helps ensure that a minimum amount of space exists between unrelated nodes.
5. **Repulsive Y Controller:** This repulsive controller operates in the vertical direction. The repulsive controller helps ensure that a minimum amount of space exists between unrelated nodes.

The XY Control algorithm begins by arranging the nodes on the circumference of a circle. Subsequently, the algorithm follows an iterative process through which the nodes are incrementally pushed towards a stable final resting spot where the forces imposed by the five controllers are in balance.

Initially, the strength of the repulsive controllers is strong in the Y direction and weak in the X direction. This allows the nodes to be moved together into a vertical column while the hierarchy is being established. Towards the end of the layout process the strength of the repulsive Y controller is reduced to allow the stable rows in the Y direction to form. The repulsive controller in the X direction is strengthened to spread out the horizontal spacing between the nodes.

The XY Control algorithm is easiest to tune when a staged process is followed. By incrementally engaging the controllers the number of constants being tuned is reduced into smaller more manageable groups. The result from the XY Control algorithm for the benchmark network is shown in Figure 7 and contains 12 avoidable edge crossings. The presentation is clearer than that obtained from the traditional Spring Embedder and Magnetic Spring algorithms.

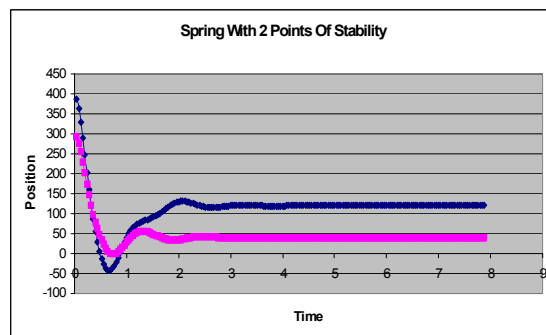
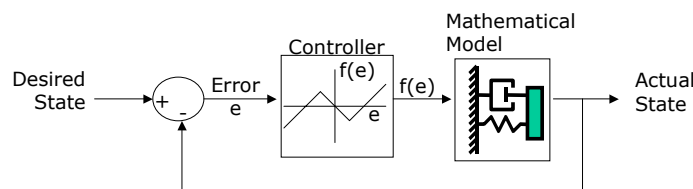


Figure 6. The Astable Controller used in the XY Control algorithm.

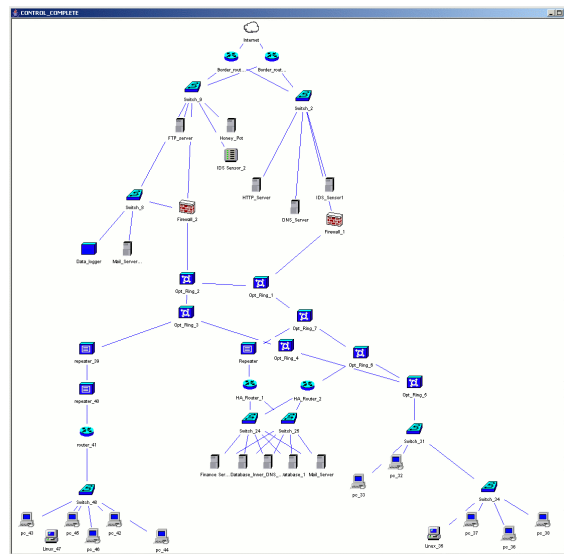


Figure 7. The XY Control algorithm applied to the benchmark network.

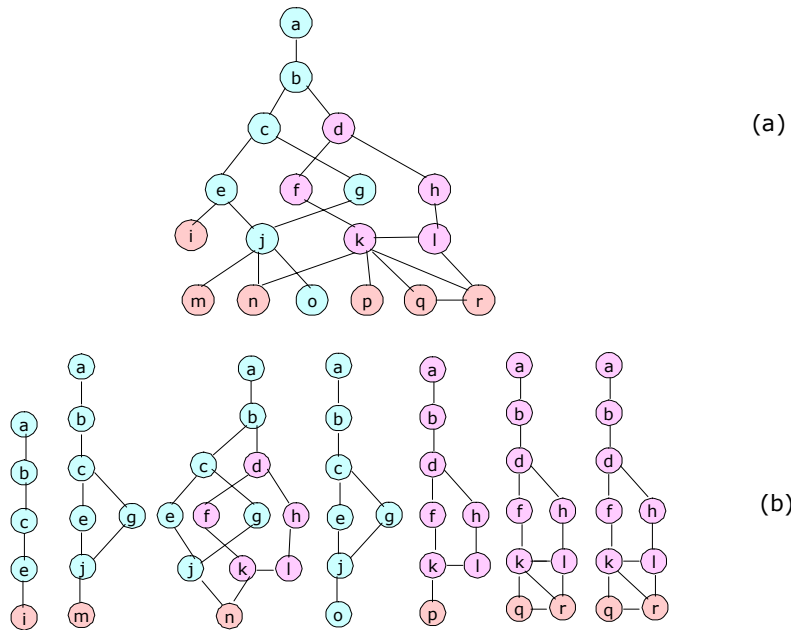
## 4.2 Voting Algorithm

The Voting algorithm is a new custom approach we designed to lay out sparsely crossed directed graphs. The algorithm optimizes both planarity and space utilization concerns in a single pass using a “divide and conquer” strategy that first simplifies the problem to its “core nodes” and then introduces a multifactor voting process to determine node placement. Once the core nodes have been placed the initial simplification steps are reversed. This entire process is illustrated with a general example that conveys the algorithm’s concepts.

The Voting algorithm begins by removing all of the end-nodes from the network diagram. An end-node is any singly-connected node that is not a root node. The remaining nodes are referred to as “core nodes”. With the end-nodes removed, the nodes that have no children<sup>2</sup> are identified. Each of these nodes marks the end of a “thread”. The threads are identified by applying a graph-walking algorithm from the end of the thread up to the root nodes. Each thread may share a common parentage with other threads but has at least one unique vertex. Figure 8 shows an example of the algorithm applied to a hypothetical network. In this example, the end-nodes have already been removed. In Figure 8(a), node (a) is the root node; nodes (i), (m), (n), (o), (p), (q) and (r) are the ends of the threads. The blue nodes in the figure highlight thread (a-o). The orange nodes are the other thread ends. Figure 8(b) shows all of the threads in the network.

<sup>2</sup> These nodes would typically be the hubs or switches near the periphery of the network.





**Figure 8.** The core nodes of a hypothetical network diagram (a) and the threads identified within it (b). The blue nodes highlight a single thread, the orange nodes denote the thread ends and the pink nodes are devices unrelated to the current thread.

In the next step, the threads are arranged in a sequence horizontally based on shared nodes such that threads having many vertices in common are placed together while unrelated threads are placed further apart. This is done by counting the number of times each node appears across threads to find the “frequency of appearance” for each node. For each thread, the sum of the nodes’ frequencies of appearance determines the “centralness” of the thread. For example, node (g) appears in 3 threads in our hypothetical network while the thread (a-o) has a centralness total of 29. The thread with the highest number of shared vertices is the central thread. The thread with the second-highest number of shared vertices is the vice thread. In the case of a tie, the one with the fewest non-shared nodes is selected as vice thread. In our example network, the central thread is thread (a-n) and the vice thread is thread (a-p). The remaining threads are then placed one at a time on the left or right hand side, depending on which side it shares the most nodes with.

Once the threads are sequenced horizontally, the nodes contained inside the layers of each thread are positioned. For thread layers that involve multiple nodes, the sequencing is based on a vote involving the four metrics that are described in Table 2.

**Table 2.** Voting Algorithm Metrics

NAME	DESCRIPTION	PRIORITY
External Vote (E-Vote)	Counts the number of external requests made against each vertex from the left and right side. A count imbalance generates a preference to migrate the associated vertex to the corresponding side.	Low
Grand-parent Vote (GP-Vote)	The vertex sequencing occurs in a top down fashion. As a consequence, the order of the parent vertices impacts the vertex order on the current level.	Medium

V Shaped Vertex Formation (V-Vote)	Children with a V-shaped connection prefer to have their parents nearby. This preference influences the proximity of the parent vertices (not their order).	High
Horizontal Vertex Formation (H-Vote)	Horizontally connected vertices should be placed together. Their order may be reversed but not re-sequenced.	Highest*

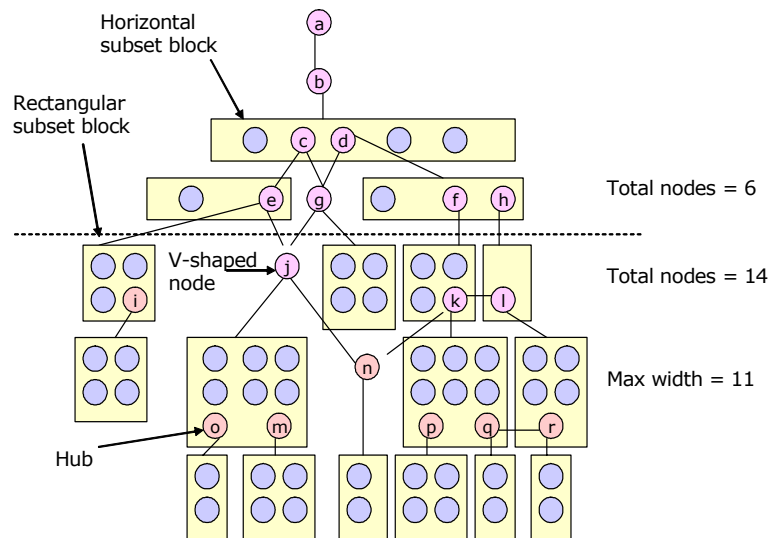
\* The highest preference rating is given to situations that frequently occur for redundant or load shared equipment which typically are placed together on a network diagram.

The first voting metric, the E-vote, biases a node's position to the left or right based on the external threads. The second metric, the GP-vote, initializes the vertex order based on past layout actions to minimize edge crossings. The third voting metric, the V-vote, acts on subgroups of vertices within a level and checks whether nodes should go together to minimize future edge crossings in the foreseeable future. The last metric, the H-vote, places laterally-connected vertices side by side.

The voting process starts at the central thread and ends at the periphery. Within each thread the process begins at the top layer of a thread and ends at the bottom layer. As the thread layers are processed, the core nodes are recombined into a single diagram. When processing a layer, nodes that have been placed previously are eliminated from the voting process. If a layer has only one remaining node to be placed, there is no voting process and the node is placed in the first open position; otherwise the voting process proceeds first with the E-vote, then the GP-vote, then the V-vote and finally the H-vote. Note that the voting metrics are applied to the thread in the order of their priority with each vote overriding the previous.

When the core nodes have been placed, the end-nodes that were removed during the first step must be reinserted. Thus far the layout process has been designed to organize the diagram based on hierarchal and planarity concerns. During end-node reinsertion, the algorithm minimizes the graph dimensions by node packing techniques. The packing strategy, demonstrated in Figure 9, is as follows:

- Nodes are placed in either horizontal or rectangular subset blocks. The horizontal subset blocks appear at the top of the diagram while the rectangular subset blocks appear at the bottom. This technique minimizes the edge routing by using a single line to connect a forwarding device to its children.
- V-shaped nodes need to be placed separately because they are members of multiple subset blocks. To minimize edge crossings, each V-shaped node is placed near the top row of a layer. If there is more than one V-shaped node interconnecting the same set of parents, the nodes are to be laid out in a vertical column.
- Forwarding devices are placed at the lowest row within a layer. The spacing between nodes within this bottom row is proportional to the number of children.

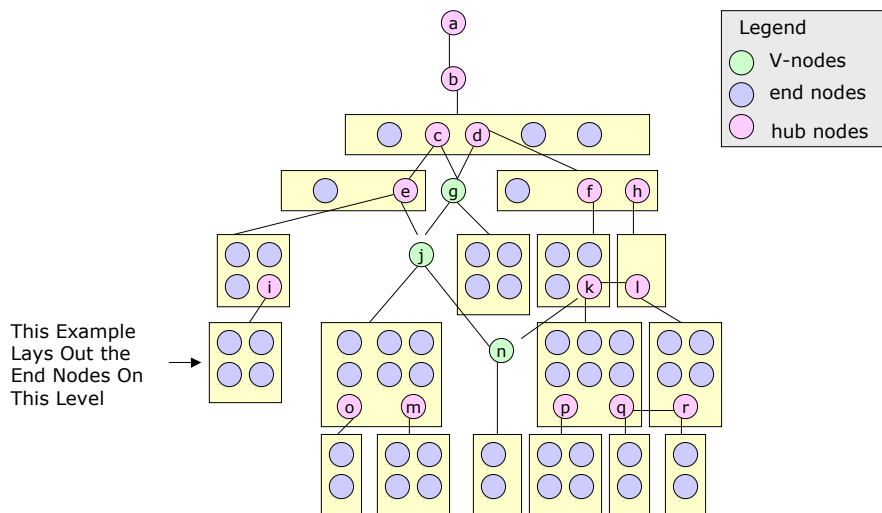


**Figure 9.** The hypothetical network showing the end node placement principles and the breakpoint between horizontal and rectangular subset blocks.

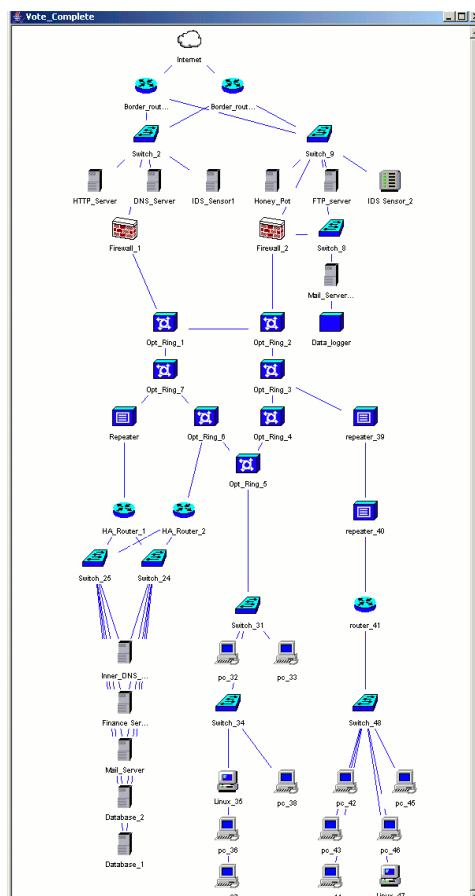
The first step in end-node placement involves determining which row will optimally act as a break point between horizontal and rectangular layers. For each level, the width and height required to optimally place the nodes in a rectangular subset is determined. This calculation begins by placing the nodes in a subset in a square, then rearranging them if needed to ensure that forwarding devices are all placed in the bottom row. Forwarding devices with many children are allocated more space (see nodes (o) and (m) in Figure 9). Any V-connected nodes are aligned in a column, which is also a determining height factor. The breakpoint between horizontal and rectangular layouts lies at the point at which the total number of nodes in a level is less than the maximum width of all the layers.

To place the horizontal subset rows, the core nodes that are left justified are centred above the lower levels. The centring process introduces extra space in the horizontal layers and usually provides the necessary openings needed to properly insert the remaining end-nodes. When there is insufficient space, a minimally disruptive node insertion algorithm is used to place the remaining nodes. The algorithm looks for a position close to its parent and away from any V-connected nodes. The completed diagram is shown in Figure 10.

The Voting algorithm was applied to the benchmark network and the results shown in Figure 11. The Voting algorithm eliminated all avoidable edge crossings and minimized graph dimensions. For the benchmark network, the Voting algorithm is slower than the DFS algorithm, but faster than the Barycenter, Median, and Sifting approaches.



**Figure 10.** The final placement of the nodes of the hypothetical network diagram using the Voting algorithm.



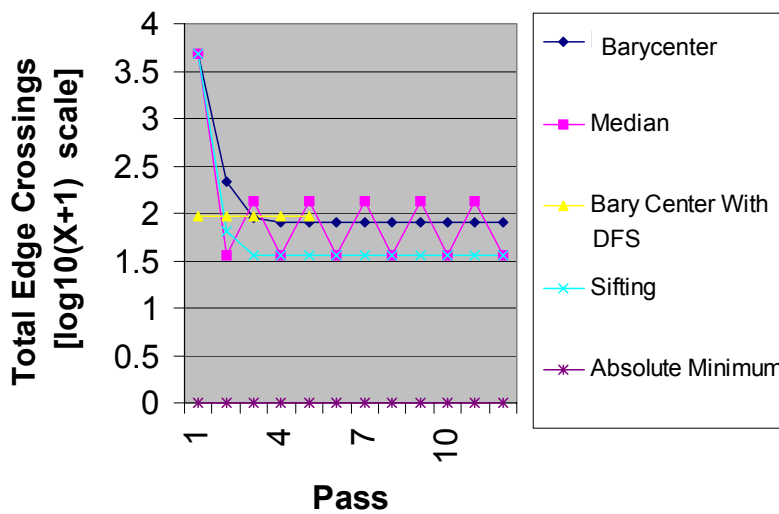
**Figure 11.** The Voting algorithm applied to the benchmark network.

## 5. EVALUATING THE RESULTS

Seven generic and two custom layout algorithms have been discussed and their output on a test network displayed. This study avoids providing exact figures for performance and planarity because many of the standard layout algorithms use an iterative approach that does not have a fixed cut-off criterion. Iterative techniques if left long enough, may eventually reduce the total edge crossing to some minimal value. As can be seen in Figure 12, the algorithms reached a relatively steady state after 5 to 10 passes for all but the Median approach, which oscillates slightly between two states. The total edge crossings for the Barycenter/Median could be further reduced by 10-20% if the algorithm were allowed to run 10 times longer. As a compromise between planarity and speed all iterative algorithms were stopped when one of the following conditions was met:

- Five consecutive up and down sweeps of the network resulted in no improvement in edge crossings (used for Barycenter, Sifting and Median algorithms)
- The total change in displacement error remains small (used for Spring Embedder, Magnetic Spring and XY Control algorithms)
- All edge crossings were eliminated (used for Barycenter, Sifting and Median algorithms)
- A maximum of X iterations had passed<sup>3</sup> (all algorithms)

It should be noted that the DFS and voting algorithms operate in a single pass.



**Figure 12.** The total number of edge crossings using the different algorithms as a function of the number of iterations. Note the logarithmic scale.

The overall evaluation was based on speed, complexity and planar quality and the results are summarized in Table 3. In terms of speed the Depth First Search is 10 times faster than the second place Voting algorithm and 1500 times faster than the last place Sifting algorithm. With the exception of the Sifting algorithm, the hierarchal directed graph approaches are faster than the force directed approaches. This means that the Median and Barycenter algorithms are noticeably faster than the Spring Embedder, Magnetic Spring and XY Control algorithms.

<sup>3</sup> X is algorithm-dependent and set to a value well within the typical steady state region. Note some layout algorithms have a tendency to oscillate. The purpose of this X is to ensure the algorithm eventually quits. The maximum iterations were set as follows Spring Embedder-2000, Magnetic Spring-2000, XY Controls-1600, BaryCenter-50, Sifting-50, and Median-50.

The number of lines of source code is the basis for judging algorithm complexity. The Depth First Search is the least complex algorithm with 19 lines of code. The Voting algorithm is most complex with 3000 lines of code. The remaining algorithms were between 200 to 500 lines in length.

**Table 3.** Comparative Evaluation of Layout Algorithms

	SPEED*	CODE COMPLEXITY**	PLANAR QUALITY***	LIMITATIONS / COMMENTS
Existing Layout Algorithms				
Depth First Search	A	A	C	Planarity improvement comes as side effect of algorithm and is not optimized.
Spring Embedder	B-	B	B	Quality of results varies on a run-to-run basis. Diagram has a characteristic firework pattern but is difficult to read.
Magnetic Spring	B-	B	B	Quality of results varies less than the Spring Embedder approach. It had the best space utilization however this was because the nodes were unevenly spaced.
Barycenter, Median, DFS with Barycenter	B+	B	B	Quality of output depends on sequence of input. Tends to get stuck in local minima because each algorithm has weakness in ability to handle parallel (or fork shaped) vertex patterns. Good for dense edge crossing situations. Median slightly better quality results than Barycenter. Prefacing Barycenter with a DFS improves solution times.
Sifting Algorithm	B-	B	B	Slightly better results than Barycenter and Median approaches but has a significant speed penalty.
New Layout Algorithms				
XY Control System	B-	B	B	Quality of results varies significantly on a run-to-run basis. Overall appearance of the diagram is much better than other force directed methods.
Voting Algorithm	A-	C	A	Generates good results for sparsely filled hierarchal directed graphs. Algorithm is very complex and difficult to debug.

\* Speed Grading System A = (<1 second), A- = (<5 second), B+ = (<30 seconds), B = (<1 minute) , B- = (<10 minute),

\*\* Code Complexity Grading System A = (<50 lines), A- = (<100 lines), B = (<500 lines), C = (<5000 lines)

\*\*\* Planar Quality A = Minimal Number Of Edge Crossings, B = Good Edge Crossing Reduction, C = Includes No Logic To Reduce Edge Crossing

The performance of a given layout algorithm is highly dependent on the code organization and underlying data structures. Small data storage inefficiencies, either intentional (i.e. to reduce memory consumption) or unintentional, can have huge impacts on algorithm speed. As an example, the initial sifting algorithm implementation took close to 8 hours but was later reorganized to run in 6 minutes.

In terms of planarity, the Voting algorithm performed best for the benchmark network because it is a sparsely crossed network. For densely crossed networks the Sifting approach generated the best diagrams<sup>4</sup>. The XY control algorithm was the best force directed approach. The Depth First Search ignored the edge crossing issue entirely and had the highest amount of planar variability.

## 6. CONCLUSIONS

Nine layout algorithms were applied to a benchmark network. The Voting algorithm, though complicated, had the best planarity with good run time performance. The Voting algorithm is not iterative, avoids computationally expensive edge crossing calculations, and is able to complete the layout process in a single pass. The algorithm uses multiple metrics to make better placement decisions and is intended for sparsely cross-connected networks.

For densely edge-crossed networks the Sifting algorithm had the best planarity but the slowest speed. For uncrossed networks the DFS algorithm is the best choice. The Median and XY Control algorithm both provide a good compromise between planarity and performance. For situations where a network hierarchy cannot be established the Spring Embedder algorithm is preferred.

The algorithms employed in this study will be used as visual aids for a tool that assesses the impact of computer security events on a network's operation and defence.

## 7. REFERENCES

- [1] Microsoft (2006) Powerpoint, <http://office.microsoft.com/en-us/FX010857971033.aspx>
- [2] Microsoft (2005) Visio 2003 Product Demo, <http://www.microsoft.com/office/visio/prodinfo/demo.msp>
- [3] Smartdraw (2005) Network Design Examples – Created With Smart Draw, <http://www.smartdraw.com/examples/networkdesign/index.htm>
- [4] NetViz, (2005) netViz Product Tour , <http://www.netviz.com>
- [5] Nimsoft (2006) NimBUS for End to End (ETE) Response Time Monitoring, <http://www.nimsoft.com/solutions/ete/index.php?trackcode=bizcom>
- [6] Computer Associates (2004), Unicenter® Network and Systems Management r3.1 [http://www3.ca.com/Files/DataSheets/unicenter\\_nsm\\_datasheet.pdf](http://www3.ca.com/Files/DataSheets/unicenter_nsm_datasheet.pdf)
- [7] Boardman, B. (2003) Network and System Management Review <http://www.nwc.com/1402/1402f23.html>
- [8] Hewlett Packard (2003) HP OpenView Network Node Manager: A Guide to Scalability and Distribution , [http://www.scd.ucar.edu/nets/docs/sysadm/openview/Scalability\\_and\\_Distribution.pdf](http://www.scd.ucar.edu/nets/docs/sysadm/openview/Scalability_and_Distribution.pdf)
- [9] Hewlett Packard (2000), HP toptools 5.6 for Unicenter, <http://docs.hp.com/en/2170/tt56tnd.pdf>

---

<sup>4</sup> If the Voting algorithm is applied to the densely cross network the entire network becomes a single thread, the E-vote becomes impotent and the remaining voting factors become overloaded and perform poorly.

- [10] Intellitactics, (2006) Solving Challenging Problems, <http://www.intellitactics.com/blue.asp?PageID=21>
- [11] OPNET (2005) [http://www.opnet.com/products/modules/flow\\_analysis.html](http://www.opnet.com/products/modules/flow_analysis.html)
- [12] Neon Software (2005) LANsurveyor Diagram and Manage Your Network, <http://www.neon.com/LSwin.shtml>
- [13] Ferrill, P. (2004) LanSurveyor 8.5 for Windows, <http://www.networkworld.com/reviews/2004/071204productpeek.html>
- [14] Adrem (2005) NetCrunch , <http://www.extralan.co.uk/products/Diagnostic-Tools/Adrem/Netcrunch.htm>
- [15] IPSwitch Inc, (2005) What's Up Professional, <http://www.ipswitch.com/products/whatsup/professional/index.asp>
- [16] Mitchell, D. (2005) Product Reviews – Security - AdRem NetCrunch Standard 3.1 <http://www.pcpro.co.uk/reviews/74204/adrem-netcrunch-standard-31.html?searchString=netcrunch+reviews+netcrunch+reviews+review+reviewed+reviewing+reviewer+reviewers>
- [17] ManageEngine (2006) OpManager, <http://manageengine.adventnet.com/products/opmanager/>
- [18] JGraph, (2005) JGraph - The Java Open Source Graph Drawing Component, <http://www.jgraph.com/jgraph.html>
- [19] Tom Sawyer Software (2006) Tom Sawyer Software <http://www.tomsawyer.com/home/index.php>
- [20] Bouchard, A (2004) Link Analysis and Visualization, DRDC Valcartier TM 2004-175
- [21] Wagner, D. (2003) Notes 3 for CS 170 , <http://www.cs.berkeley.edu/~daw/teaching/cs170-s03/Notes/lecture3.pdf>
- [22] Eades, P. (1984) A Heuristic for Graph Drawing, *Congressus Numerantium*, **42**, 149-160
- [23] Sugiyama, (2002) Graph Drawing and Applications For Software and Knowledge Engineers, World Scientific Publishing Co. Pte. Ltd. pages 19-101
- [24] Sugiyama, K; Tagawa, S; Toda, M. (1981) Methods for Visual Understanding of Hierarchical System Structures, *IEEE Transactions Systems, Man, And Cybernetics*. **SMC-11**, (2), 109-125 [http://plg.uwaterloo.ca/~itbowman/CS746G/Notes/Sugiyama1981\\_MVU/](http://plg.uwaterloo.ca/~itbowman/CS746G/Notes/Sugiyama1981_MVU/)
- [25] Stallmann, M., Brglez, F., and Ghosh, D. (1998) Heuristics and Experimental Design for Bigraph Crossing Number Minimization, *Proceedings of the First Workshop on Algorithm Engineering and Experimentation*, ALENEX '99, Baltimore, MD, USA, Jan. 15-16. pp. 74-93., <http://citeseer.ist.psu.edu/15341.html>
- [26] Koutsofios, E. (1996) Drawing graphs with dot, Technical report, AT&T Bell Laboratories, Murray Hill, NJ. <http://www.research.att.com/sw/tools/graphviz/dotguide.pdf>



- [27] Gansner, E., Koutsofios, E., Notrth S., and Vo, K. (1993) A Technique For Drawing Directed Graphs, *IEEE Transactions on Software Engineering*, **19** (3), 214-230  
[http://www.research.att.com/areas/visualization/papers\\_videos/papers/1993gknv.pdf](http://www.research.att.com/areas/visualization/papers_videos/papers/1993gknv.pdf)
- [28] Eades, P. (1994) Edge Crossings in Drawings of Bipartite Graphs, *Algorithmica*, **11**, 379-403
- [29] Matuszewski, C. (1999) Using sifting for k-layer straightline crossing minimization , *Proceedings of the 7th International Symposium on Graph Drawing*, pages 217-224, <http://www.mathematik.uni-halle.de/reports/shadows/99-11report.html>
- [30] Rudell, R. (1993) Dynamic Variable Ordering for Ordered Binary Decision Diagrams, *Proceedings International Conference on Computer Aided Design (IC-CAD)*, pages 42-47
- [31] Battista, G. et al. (1994) Algorithms for Drawing Graphs and Annotated Bibliography, *Computational Geometry: Theory and Applications*, **4** (5). 235-282.
- [32] Herman, I. (2000) Graph Visualization and Navigation in Information Visualization: a Survey, *IEEE Transactions on Visualization and Computer Graphics*, **6** (1), 24—44  
<http://homepages.cwi.nl/~ivan/AboutMe/Publications/StarGraphVisuInInfoVis.pdf>
- [33] Stallmann, M., Brglez, F., and Ghosh, D. (2001) Heuristics, Experimental Subjects, and Treatment Evaluation in Bigraph Crossing Minimization, *Journal of Experimental Algorithmics* 6, 8.,  
<http://citeseer.ist.psu.edu/311042.html>

