Finding Weak Points in Networks

George Dean Bissias, Brian Neil Levine, and Ramesh K. Sitaraman
Dept. of Computer Science, Univ. of Massachusetts Amherst, USA
{gbiss,brian,ramesh}@cs.umass.edu

Abstract

Client-server networks are pervasive, fundamental, and include key networks such as the Internet, power grids, and road networks. In a client-server network, clients obtain a service by connecting to one of a redundant set of servers. These networks are vulnerable to vertex and link failures as some clients may become disconnected from the servers causing them denial of service. We develop algorithms that quantity and bound the inherent vulnerability of a client-server network using semidefinite programming (SDP) and branch-and-cut techniques. Further, we develop a divide-and-conquer algorithm that solves the problem for large graphs. We use these techniques to show that: for the Philippine Power Grid removing just over 6% of the transmission lines will disconnect at least 20% but not more than 50% of the substations from all generators; on a large wireless mesh network disrupting 5% of wireless links between relays removes Internet access for half the relays; even after any 16% of Tier 2 ASes are removed, more than 50% of the remaining Tier 2 ASes will remain connected to the Tier 1 backbone; when 300 roadblocks are erected in Michigan, it is possible to disconnect between 28–43% total population from all airports.

1 Introduction

A client-server network consists of clients and servers where each client obtains a critical service by connecting over the network to a server chosen from a redundant set of servers. The key aspects of several real-world networks can be modeled as client-server networks, with suitable definitions for clients and servers. For instance, a wireless mesh consists of mobile users (i.e., clients) accessing the wired Internet by connecting to one of the gateways (i.e., servers). In the Internet context, Tier-2 autonomous systems (i.e., clients) connect to the Internet backbone by peering with one of the Tier-1 autonomous systems (i.e., servers). In a power grid, each substation (i.e., client) connects to one of the power generators (i.e., servers) to obtain electricity. And, people (i.e., clients) travel to airports and other critical service infrastructure (i.e., servers) using the road network.

Client-server networks are vulnerable to vertex and link failures as clients that are unable to connect to any server are denied service. Failures may be caused by both genuine malfunction and intentionally targeted attacks. Even though servers are typically deployed in a distributed and redundant fashion to enhance service availability, a targeted set of failures can cause serious denial of service to clients. Network attacks and failures are common in a wide variety of networks. Wireless networks are susceptible to numerous attacks including DOS, spoofing attacks, and physical tampering [20, 23, 31, 32]. In wired networks, fiber optic cables and other media are conspicuously susceptible to intentional or inadvertent destruction [8, 16, 24]. Internet Autonomous Systems are vulnerable to BGP protocol exploits that have been documented by numerous researchers for years [19,22,25,27]. Power grids are crucial to modern infrastructure as illustrated by the major blackout in North America in 2003 and many point out targeted (terrorist) attacks are a significant concern [6, 7, 29]. The industry consensus is a need to identify attack and failure scenarios before they happen [6, 28]. Finally, there is significant concerned for the national highway system. A recent study identified eight unrelated suspicious events that seemed to indicate terrorist attention to major pieces of domestic highway infrastructure [18] via roadside improvised explosive devices (IEDs) or vehicle-borne IEDs. To address these concerns, the United States Government Accountability Office calls for individual vulnerability assessments on the nationally critical Tier-2 structures list [10].

All of the above examples of real-world client-server networks underscore the importance of studying the effect of failures and attacks on service availability for clients. Our specific goal is to assess the intrinsic vulnerability of a client-server network by computing a quantitative relationship between the number of worst-case attacks and failures and the number of clients who are denied service.

1.1 Problem Statement

A client-server network can be modeled as a graph $G = (S \cup C, E)$, where S is a set of server vertices, C is a set of client vertices, and E is the set of edges. Each server is equally capable of providing the required service and each client accesses the service by connecting to any one of the servers in S. Taking the power grid as an example, the set S consists of generator nodes that serve electricity and the set C consists of substation nodes that receive and distribute it to customers. If any substation is disconnected from *all* generators, then it and all of its customers will lose power. Wireless mesh networks can also be modeled as client-server graphs where set S is the small number of *gateways* that are directly connected to the Internet, while the set C consists of the large number of *relays* that connect to gateways or other relays. Internet connectivity is established for any given relay only when there exists some path to a gateway.

Our problem can be formally defined as follows. We define a *service path* for a client vertex v as any path in G that begins at v and ends at some server vertex. Client v is said to be *serviceable* if and only if it has a service path. Let $Conn(G)$ be the number of serviceable clients in G. Let $G \ominus R$ be the graph G with elements $R \subseteq S \cup C \cup E$ removed, i.e., $G \ominus R$ is the graph that results when

the vertices and edges in R fail[1]. We focus on solving the following problem in the context of real networks.

PROBLEM: $MinConn(G, n, m)$

- Given: client-server graph $G = (S \cup C, E)$ and $n, m \in \mathbb{R}$.
- Find: minimum value of $Conn(G \ominus (A \cup B))$, over all A, B, where $A \subseteq C, |A| = n$, and $B \subseteq E, |B| = m$.

Note that $MinConn(G, n, m)$ is simply the number of serviceable clients in G in the *worst* scenario of n client vertex failures and m edge failures.

We also solve a weighted version of the above problem that we call *WeightedMinConn*(G, n, m) where each $v \in C$ is assigned a weight $w[v]$. The objective is to minimize the weighted sum of the serviceable clients in $G \ominus (A \cup B)$, where $A \subseteq C$ and $B \subseteq E$, with $\sum_{v \in A} w[v] \leq n$ and $|B| = m$. The weighted variant of the problem is useful in our road network analysis where each vertex represents a geographical area and is weighted by its population density.

1.2 Our Results

Our main contributions are as follows. We formalize the notion of vulnerability in client-server networks by defining the *MinConn* problem and its weighted variant. Since solving the *MinConn* problem exactly is NP-Hard (see Section 4), we develop algorithms that derive upper and lower bounds. We provide the first efficient algorithms for bounding *MinConn* for both edge and client vertex removal (separate or combined) using semidefinite programming and a branch-and-cut technique. Further, we extend these techniques to large graphs using a divide-and-conquer technique. Finally, we demonstrate the wide applicability of our techniques on real-world data sets, providing quantitative results.

Our real-world analysis includes *(i)* a residential wireless mesh network, *(ii)* the power grid of the Philippines, *(iii)* the Internet AS level graph, and *(iv)* the highway systems in Iowa and Michigan. We chose these datasets for their importance, diversity, and size. In particular, the AS and highway system graphs are large at over 5,000 and 1,500 vertices, respectively.

For the mesh network, roughly half the relays are disconnected from all gateways after removing just 12 links ($< 5\%$). Direct relay removal is less dramatic but more damaging for this mesh. We show that removing 12 relays ($< 18\%$) can disconnect more than 70% of the remaining relays. Similarly, for the Philippine power grid, removing just over 6% of the transmission lines will disconnect at least 20% but not more than 50% of the substations from all generators. On the other hand, we also find that almost 50% of the substations can be disconnected from all generators after strategically removing a certain 20 (approximately 5%) other substations.

[1] We assume that failed vertices and links cannot be used for any purpose, and hence may be removed.

To evaluate the two remaining data sets, we use our algorithm for large graphs. The Autonomous Systems (AS) level graph of the Internet remains resilient, as we show that even when 900 ($> 16\%$) Tier-2 nodes are removed, more than half the Internet remains connected via the Tier-1 backbone. We also used this approach to analyze the number of people with highway access to airports in the states of Iowa and Michigan. We show that when 200 roadblocks are erected state-wide it is possible to disconnect between 500K and 1.7M people (15–50% total population) from every airport in Iowa. In Michigan we found that only 2.8M to 4.3M people (28–43% total population) will maintain airport connectivity after erecting 300 road blocks.

1.3 Related Work

A traditional approach to studying the vulnerability of a complex network is to use graph partitioning techniques such as finding graph separators of small width [26]. An α-separator of a graph $G = (V, E)$ is a collection of either edges or vertices whose removal separates a graph into two disconnected subgraphs, each having size at most $\lceil (1 - \alpha)|V| \rceil$. The number of edges or vertices in the cut is called the separation *width*. Finding either an α-edge-separator or α-vertex-separator of minimum width is NP-hard. Bui and Jones have shown it is NP-hard even to find a good approximation to the α-separator problem [11]. The objective of graph separation is to completely partition the graph into appropriately-sized pieces by removing vertices and edges. In contrast, our goal is not to partition the graph completely, but to study the manner in which service availability degrades as a function of the number of failed components.

Even though standard graph partitioning does not appear to be directly applicable, some of the techniques developed in that context have been key to our approach. Specifically, Wolkowicz and Zhao [30] developed techniques for using semidefinite programming for graph partitioning. We utilize some of those ideas in our algorithm, specifically the notion of a lift matrix to represent our block decomposition of the client-server graph.

Numerous works attempt to experimentally evaluate the vulnerability of specific networks or graph families. However, they do not consider a client-server formulation with a focus on service availability as we do here. Neither do they develop algorithms that analyze worst-case failure scenarios as in our work. Ding et al. [14] used spectral techniques to decompose the web graph. Magoni and Zhou et al. [21, 33] independently analyzed the Internet AS graph and showed that by removing approximately 3% of the ASes (including Tier-1's and Tier-2's) the size of the largest connected component decreases by a factor of 3. Flaxman et al. [15] provides a framework for strengthening existing graphs by supplementing them with regular subgraphs. Cohen et al. [12] provided a modification to classical percolation theory in order to study the effects of removing vertices with highest degree, and demonstrated that even though scale-free graphs are robust to random failure they are highly sensitive to targeted attack.

1.4 Roadmap

In Section 2, we provide an algorithm that uses semidefinite programming (SDP) and branch-and-cut techniques to bound the values of $MinConn(G, n, m)$ for arbitrary client-server networks. Further, we apply these techniques to assess the vulnerability of two real-world networks: a wireless mesh network and the power grid of the Philippines. In Section 3, we extend the SDP-based approach to larger graphs by using a divide-and-conquer heuristic. Further, we apply this approach to two larger real-world networks: the Internet AS graph, and the road networks of Iowa and Michigan. Finally, in Section 5, we offer concluding remarks.

2 An SDP-based Algorithm

The goal of this section is to derive upper and lower bounds on $MinConn(G, n, m)$ by means of a semidefinite program (SDP). For clarity, the SDP is translated from a conceptually simple, but numerically unstable, Quadratically Constrained Quadratic Program (QCQP) formulation. Fundamental to our approach is a *block* representation for G that separates serviceable, non-serviceable, and entirely removed clients.

2.1 QCQP Formulation

Let A be the adjacency matrix of $G = (S \cup C, E)$, and for convenience label $V \equiv S \cup C$. Further, for any $V' \subseteq V$, let $G(V')$ denote the induced subgraph of G whose vertex set is V' and whose edge set is the subset of E consisting of those edges with both ends in V'. A *block* B is any (possibly empty) subset of V such that $G(B)$ is disconnected from $G(V \setminus B)$, but neither subgraph $G(B)$ nor $G(V \setminus B)$ are necessarily connected individually. We designate three blocks B_1, B_2, and B_3, with $|B_i| = b_i$. The *block indicator matrix* for G is a $|V| \times 3$ matrix X, where

$$X(i,j) = \begin{cases} 1, & \text{node } i \in B_j \\ 0, & \text{otherwise} \end{cases} . \tag{1}$$

It follows that $\sum_i X(i,:) = 1$ and $\sum_j X(:,j) = b_j$, where ":" denotes any cell along that dimension (as in Matlab or Python).

We next seek to frame the *MinConn* problem in terms of the three blocks we have created above, illustrated in Fig. 1. These blocks will correspond to a segmentation of serviceable, non-serviceable, and removed clients in G. Specifically, the first block is the server block, to which all server vertices are fixed, and clients with paths to these servers are placed. Clients without paths to servers are placed in the second block. Finally, the third block is the *null* block, which is where clients that have been removed from the graph are placed. The set of removed vertices and edges are called the vertex and edge *cuts*, and they are labeled $\mathcal{C}_V$ and $\mathcal{C}_E$ respectively. We have the following QCQP formulation.

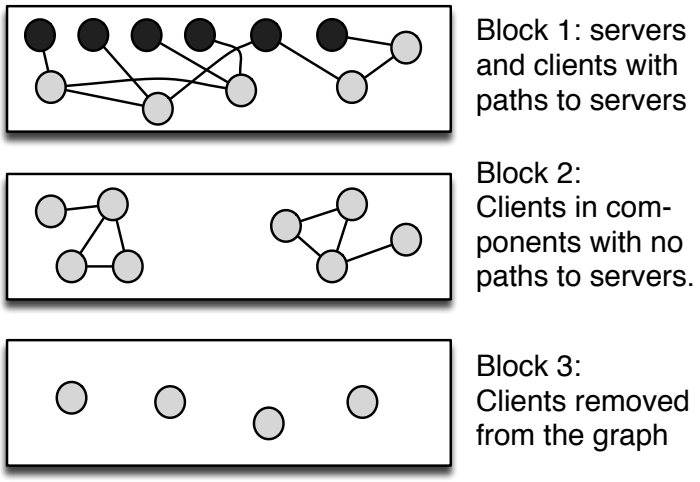


Figure 1: Separating a Client-Server Graph into blocks

- Given: client-server graph $G = (S \cup C, E)$ and $n, m \in \mathbb{R}$.
- Find: minimum $\| X(:,1) \|$ such that,

(1) $X(i,1) = 1, \ \forall i \in S.$ (All servers in Block 1)

(2) $\sum_{i=1}^{|V|} \sum_{j=1}^{3} X(i,j) \leq |V|.$ (Vertex Count)

(3) $|\mathcal{C}_V| \leq n.$ (Vertex Cut Size)

(4) $|\mathcal{C}_E| \leq m.$ (Edge Cut Size)

(5) $X(i,j) \in \{0,1\}$ (0–1 property)

Note that $\mathcal{C}_V \equiv B_3$ so

$$|\mathcal{C}_V| = \sum_{i=1}^{|V|} X(:,3). \tag{2}$$

Finding an expression for $|\mathcal{C}_E|$ is just a bit more complicated. We begin with a short digression. Vector $X(:,j)$ indicates the vertex membership of block j. That is to say, $X(i,j) = 1$ if vertex i is in block j and is 0 otherwise. For an arbitrary matrix M, the quantity $X(:,j)^T M X(:,j)$ gives the sum of entries in the sub-matrix indicated by $X(:,j)$. Define D to be the matrix with row sums of adjacency matrix A along its diagonal, and further define the *Laplacian* matrix $L = D - A$. The quantity $X(:,j)^T D X(:,j)$ gives twice the *total* number of edges incident to vertices indicated by $X(:,j)$, and $X(:,j)^T A X(:,j)$ gives twice the number of edges *fully contained* in block j. It follows that $X(:,j)^T L X(:,j)$ gives twice the number of edges incident to, but not fully contained in, block j.

To quantify $|\mathcal{C}_E|$, we seek to bound the number of edges between blocks 1 and 2, that is, all edges connecting the two blocks. This quantity is captured by

the following

$$|\mathcal{C}_E| = \frac{1}{2}\sum_{j=1}^{2} X(:,j)^T L X(:,j) - \frac{1}{2} X(:,3)^T L X(:,3). \tag{3}$$

The first term gives the total number of edges that lie between *any* of the three blocks. The second term subtracts the number of edges that pass between the null block and the others.

2.2 SDP Formulation

The QCQP formulation cannot be solved directly because there is no mechanism to enforce orthogonality between block vectors; hence there is no way to force each vertex into a single partition. Directly enforcing orthogonality is not possible in a QCQP. Therefore, we must graduate to a richer language, that of semidefinite programming. Every semidefinite program with an $N \times N$ constraint matrix can be solved in time $O(N^3)$ [17]. A semidefinite program is any problem of the form

Minimize $C \bullet Z$ **subject** to $A \bullet Z = b$ and $Z \succeq 0$,

where $A \bullet B = trace(A^T B) = \sum_{ij} A_{ij} B_{ij}$ is the *Frobenius inner product* and $Z \succeq 0$ indicates that Z is positive semidefinite.

The Frobenius inner product gives the component-wise product of two matrices. So, for example, the ij^{th} entry of $A \bullet Z$ is equal to the product of the ij^{th} entry of A and the ij^{th} entry of Z. Our goal is to implement each of the constraints in the QCQP formulation in terms of a semidefinite optimization variable. To that end, let vec(X) be the vector formed by concatenating the columns of X. Furthermore, let

$$y = \begin{bmatrix} 1 \\ \text{vec}(X) \end{bmatrix}. \tag{4}$$

Define the *lift matrix* associated with X as $Y = yy^T$. Wolcowicz and Zhao [30] were the first to use this representation in the context of graph decomposition. First order terms are found along the first row and column, while second order terms are easily extracted from the remainder of the matrix. For notational simplicity, we number the rows and columns of Y beginning with index zero. We will also use the function $\phi(i,j) = i + m(j-1)$, which maps the entry $X(i,j)$ to its corresponding entry in vec$(X(i,j))$. The matrix Y comprises all pairwise products of block indices of the form $X(i_1,j_1)X(i_2,j_2)$, as well as single indices $X(i,j)$. Term $X(i_1,j_1)X(i_2,j_2)$ indicates that vertex i_1 appears in block j_1 *and* vertex i_2 appears in block j_2. We have,

$$Y(\phi(i_1,j_1),\phi(i_2,j_2)) = X(i_1,j_1)X(i_2,j_2),$$

while

$$Y(0,\phi(i_1,j_1)) = Y(\phi(i_1,j_1),0) = X(i_1,j_1).$$

Now suppose that $B(\phi(i_1, j_1), \phi(i_2, j_2)) = \beta$ for some matrix B, then the term $\beta X(i_1, j_1)X(i_2, j_2)$ will appear in row $\phi(i_1, j_1)$ and column $\phi(i_2, j_2)$ of matrix $B \bullet Y$. Constraints can thus be built as scaled single and pairwise products of block indices.

THEOREM: There exists an SDP whose solution is at most the solution of our QCQP formulation.

PROOF: It will suffice to rewrite the objective and each of the constraints of our QCQP formulation as an SDP in terms of the lift matrix Y, where Y is relaxed to the space of all semidefinite matrices. To that end let $\nu \equiv |V|$, and define $1_{i\times j}$ and $0_{i\times j}$ to be the matrices of all ones and zeros, respectively, and having size $i \times j$. The operator *diag* is defined as in Matlab so that it either extracts the diagonal of a matrix or forms a diagonal matrix from a vector. Also, let $I_{i\times i}$ be the identity matrix of size $i \times i$ and the symbol "$\otimes$" denote the Kronecker Product of two matrices. The following is an SDP formulation.

PROGRAM: *MinConnSDP*(G, n, m)

- Given: graph $G = (S \cup C, E)$ and $n, m \in \mathbb{R}$.
- Find: minimum $\mathcal{O} \bullet Y$ such that,

(1) $\mathcal{F}_{i1} \bullet Y, \ \forall i \in S.$ (Servers in Block 1)

(2) $\mathcal{S} \bullet Y = \nu.$ (Vertex Count)

(3) $\mathcal{V} \bullet Y = n.$ (Vertex Cut Size)

(4) $\mathcal{E} \bullet Y = m.$ (Edge Cut Size)

(5) $\mathcal{A}_{ij} \bullet Y, \ \forall i, j.$ (0–1 property)

(6) $\mathcal{U} \bullet Y = 1.$ **(Upper Left Equal 1)**

(7) $\mathcal{Z}_i \bullet Y, \ \forall i.$ **(Sum Vertex Values is 1)**

where

- $\mathcal{O} = diag(\begin{bmatrix} 0 & 1_{1\times\nu} & 0_{1\times 2\nu} \end{bmatrix})$
- $\mathcal{S} = diag(1_{|y|\times|y|})$ except that $\mathcal{S}(0,0) = 0$
- $\mathcal{V} = diag(\begin{bmatrix} 0 & 0_{1\times 2\nu} & 1_{1\times\nu} \end{bmatrix})$
- $\mathcal{E} = \begin{bmatrix} 0 & 0_{1\times 2\nu} & 0_{1\times\nu} \\ 0_{2\nu\times 1} & \frac{1}{2}I_{2\times 2} \otimes L & 0_{2\nu\times\nu} \\ 0_{\nu\times\nu} & 0_{\nu\times 2\nu} & -\frac{1}{2}L \end{bmatrix}$
- $\mathcal{A}_{ij} = 0_{|y|\times|y|}$ except that $A_{ij}(\phi(i,j), 0) = 1$ and $A_{ij}(\phi(i,j), \phi(i,j)) = -1$
- $\mathcal{F}_{ij} = 0_{|y|\times|y|}$ except that $F_{ij}(\phi(i,j), \phi(i,j)) = 1$
- $\mathcal{U} = diag(1_{|y|\times|y|})$ except that $\mathcal{U}(0,0) = 1$
- $\mathcal{Z}_i = 0_{|y|\times|y|}$ except that $Z_i(\phi(i,j), \phi(i,j)) = 1$ for all $j \in \{1, 2, 3\}$

Constraints for the edge cut and the 0-1 property were first introduced by Wolcowicz and Zhao [30], but we have modified the edge cut constraint to

subtract the number of edges passing to the null block. The last two constraints, in bold, are new; we have introduced them for the purposes of numerical stability. When translating constraints for mathematical programs, it is often necessary to use decompositions that involve simpler and sometimes redundant (in the space of binary vectors) constraints.

□

2.3 Solving the SDP Formulation

Semidefinite Programs can be solved by standard software suites. We use SDPT3 for MATLAB. This yields a tight lower bound for very small graphs. However, as the size of the input graph increases, the solver fails to assign each vertex to a single block. Instead, vertices are fractionally assigned to all blocks. We address this issue by fixing a small number of vertices to a single block as part of a branch-and-cut algorithm. Our entire solution procedure is as follows.

PROCEDURE: *MinConnBC*(G, n, m)

- Solve Program *MinConnSDP*(G, n, m) using SDPT3.
- Use branch-and-cut procedure to find a lower bound.
- Derive Upper bound by rounding.

The next section provides the details of this approach.

2.3.1 Lower Bound via Branch and Cut

Let $\mathcal{S}$ be the *value* of the semidefinite programming solution from Section 2.2. $\mathcal{S}$ offers a lower bound on the optimal solution to *MinConn*(G, n, m). This bound can be strengthened by fixing a small set of vertices T, each to one of the three blocks. In doing so, we move the semidefinite variable Y closer to being a binary matrix, which shrinks the search space and ultimately leads to a tighter bound. On the other hand, we cannot be certain that the resulting solution is optimal because it's possible that the optimal solution does not admit the placement we have chosen. However, if we evaluate *all* $|Y|^3$ possible permutations of vertices T, then we can be certain that the lowest solution among them is no greater than optimal.

We choose the vertices of T one-by-one. The first vertex, v_0, is chosen to be that whose placement is most ambivalent in $\mathcal{S}$ — that is, closet to one-third in each block. For each branch on v_0, the next vertex, v_1, is chosen as the most ambivalent vertex assignment in the given branch. This process is continued until the entire set T is formed. Any given vertex i is fixed to block j by adding the constraint $\mathcal{F}_{ij} \bullet Y = 1$ to the SDP. To stem the exponential growth of solution branches, we can leverage the structure of the graph G itself to prune some of these branches. First, we can eliminate any branch that places more than m pairs of connected vertices in different blocks (not including the null block). Second, we can cut any branch that places a vertex i in block 2 when i is adjacent to some server.

2.3.2 Upper Bound via Rounding

Our goal is to round some solution matrix Y^* from the solution tree formed via branch-and-cut into a concrete solution to Problem $MinConn(G, n, m)$. In principle, any partial solution from the tree is a valid candidate, but in practice we find it difficult to round most solutions while simultaneously keeping the objective value low and satisfying the edge cut constraint. So we try multiple solution matrices and choose the best rounded solution among them. The best candidates are those matrices Y^* that are nearly binary, and for small graphs these are readily found in the solution tree. However, for large graphs, a binary solution does not emerge immediately, so we choose the best branches at the last level of the solution tree and follow them exclusively until a nearly binary matrix Y^* is found.

We now proceed under the assumption that a suitable matrix Y^* has been identified. Given Y^*, define Z^* to be the block matrix formed by extracting the diagonal of Y^* and being fashioned in a manner analogous to the block matrix X. Matrix Z^* is therefore an $|V| \times 3$ matrix with each row corresponding to a single vertex and possessing net weight 1. Entry $Z^*(i, b)$ indicates the presence of vertex i in block b, which may be fractional. Rounding Z^* means rounding each row so that a single column in that row is equal to 1. This constitutes an unambiguous assignment for each vertex.

Finally, we apply the Kernighan-Lin Algorithm [26] to reduce the number of edges in the cut. The the Kernighan-Lin Algorithm is an iterative procedure that begins by swapping every possible pair of vertices (v_0, v_1) one-at-a-time where v_1 is in block 1 and v_2 is in block 2. The pair that lowers the edge cut by the most is chosen and the vertices in the pair are *fixed* to their respective blocks. This procedure is repeated until there are no unfixed vertices remaining in one or both blocks. This procedure is attractive because it preserves the objective value and is known to produce small edge cuts when the initial partition is a good one.

2.4 Empirical Evaluation

We compared the lower and upper bounds found by executing Procedure *MinConnBC*(G, n, m) in two different real world networks: a wireless mesh network from the Net Equality Project and the Philippine Power Grid.

2.4.1 Net Equality Wireless Mesh Network

In the context of a wireless mesh network, vertex removal implies that a relay has been disabled while link removal implies that the connection between two nodes has been disrupted. Here, a relay node is a client and gateway node a server; our trace had no information about the number of users (laptops, desktops, etc.) connected to clients. We investigated how the number of serviceable clients decreases as other clients and links are removed from the network.

The Net Equality Project [4] works to provide Internet access to low-income communities where residents live in dwellings within close proximity, such as

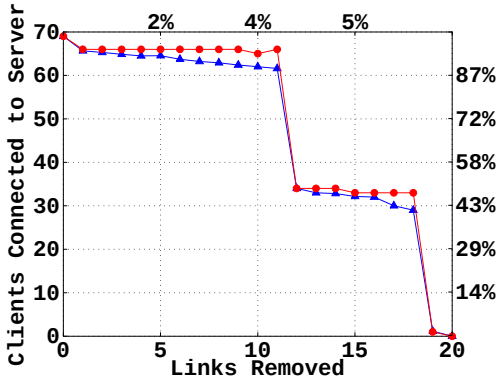


Figure 2: Wireless mesh connectivity (link removal)

apartment complexes. They deploy a small number of gateway nodes that provide direct access to the Internet and relatively inexpensive Meraki Mini access points (relays) around the community, where most relays are multiple hops from gateways. A resident has access to the Internet wherever he/she is in range of either a gateway or a relay that is ultimately connected to a gateway. We analyzed a snapshot of the Hacienda CDC housing development taken in 2007. At the time, 69 relays and 2 gateways were connected by 279 links serving approximately 1,200 residents. We studied the removal of up to 18 clients and 20 links in the Net Equality Network.

Figure 2 shows the degradation of client serviceability in the network as links are removed. The blue triangles give a lower bound on the number of relays connected to the Internet after the indicated number links have been removed. Circles in red provide a complimentary upper bound on the number of serviceable clients. These values correspond to partial (lower bound) and complete (upper bound) block assignments for each vertex. Two major features dominate Figure 2, the first is a severe drop in client serviceability after allowing 12 links to be removed. The second is another drop when 19 links can be removed. These points coincide with events where large subgraphs are finally disconnected from both gateways. This behavior provides valuable insight into the fault tolerance and attack vulnerability of the Net Equality graph. Communication difficulties between a handful of relays will not usually cause a major disruption, but degradation is not smooth, in general.

Figure 3 offers a view of client removal. Again, triangles in blue indicate a lower bound on the number of remaining clients that are serviceable when the given number of clients have been removed. Circles in red offer a complimentary upper bound. In contrast to the link removal problem, client removal exhibits a much more uniform decline in serviceability. However, while the rate of deterioration is roughly uniform, it is much more severe for the same number clients removed as links. For example, removing 8 clients can disconnect as many as 39 clients from all servers, while removing 8 links will disconnect no more

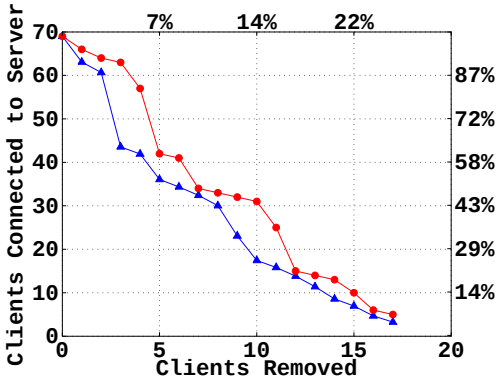


Figure 3: Wireless mesh conectivity (relay removal)

than 10 clients.

2.4.2 Philippine Power Grid

We also analyzed a network representing the Power Grid of the Philippines from 2006 [5]. The Grid comprised 44 power generators and 360 substations connected by 796 transmission lines. We modeled the Grid as a client-server graph with power generators corresponding to servers, substations corresponding to clients, and transmission lines corresponding to edges. We studied, independently, the effect of removing up to 50 transmission lines (links) or 50 substations (clients).

Figure 4 shows how power service degrades as the number of links removed increases. The lower bound departs somewhat from the upper, but does serve to establish that removing as many as 50 of transmission lines will not disconnect more than 50% of the substations. On the other hand, the plot also indicates that it *is* possible to disconnect roughly 20% of the substations after removing some 50 transmission lines.

Figure 5 shows client serviceability degradation with client removal. The most noticeable feature is the greater impact client removal has on client serviceability compared with link removal. The bounds also depart from each other more gradually than in the case of link removal. The figure shows that almost 50% of the substations can be disconnected by strategically removing just 20 other substations. In other words, removing fewer than 5% of the substations can leave half the remaining grid without service.

2.5 Topological Influence on Serviceability

Removing one client is always at least a damaging to serviceability as removing one link, so it makes sense that client removal is more damaging than link removal in both the Wireless Mesh and Power Grid networks. But the Wireless Mesh Network exhibits profound robustness even after a relatively large quantity

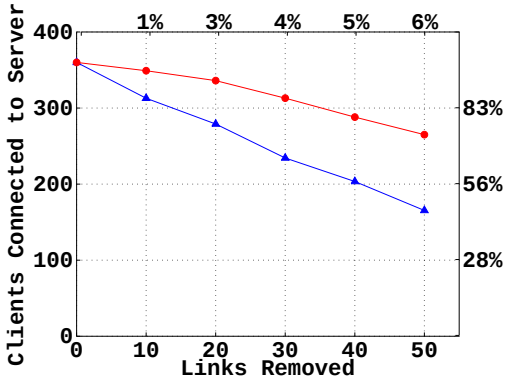


Figure 4: Power Grid connectivity (link removal)

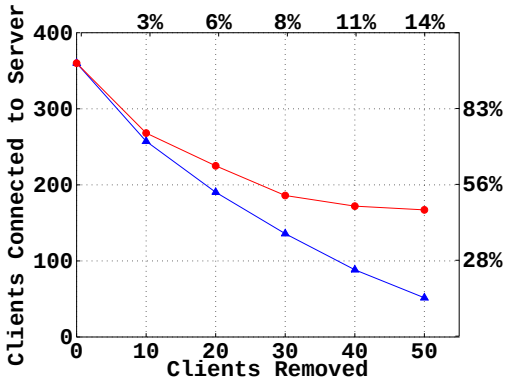


Figure 5: Power Grid connectivity (substation removal)

of links have been removed while the Power Grid does not. Upon examination we find that clients in the former have a much more rich degree distribution than clients in the later. Even though the Wireless Mesh Network has just 20% of the clients as the Power grid, more than half the clients in the Wireless Mesh network share a link with at least eight other nodes. In contrast, more than half the nodes in the Power Grid share four or fewer links with adjacent nodes. We hypothesize that it is this dramatic difference in degree distribution between the two networks that develops such a significant difference in robustness.

3 An Algorithm for Large Graphs

For large graphs, even the branch-and-cut procedure *MinConnBC* will fail because the main component *MinConnSDP* itself either fails to converge or returns an extremely weak bound. In this section, we provide an additional solution to bound $MinConn(G, n, 0)$ that instead uses *MinConnSDP* as a subroutine in a

divide-and-conquer algorithm. Our approach is the first capable of providing both upper and lower bounds on connectivity after vertex removal in graphs with thousands of vertices. The limitation of this extension is that our analysis does not apply to edge cuts and it will not work well for every graph. Success is very topologically dependent.

3.1 A Divide-and-Conquer Solution for Client Removal

Let G be a client-server graph. Our strategy is to solve $MinConn(G, n, 0)$ by breaking G into subgraphs $G_1, \ldots, G_n$ by groups of clients that act as gateways for other clients. The connectivity properties of subgraph G_i can be found by solving a series of subproblems of the form $MinConnBC(G_i, n_i, 0)$. Once the connectivity properties of each subgraph G_i have been quantified, the solution to $MinConn(G, n, 0)$ is formed by judiciously choosing those subproblems that render the lowest client-server connectivity.

Our objective is to count the minimum number of clients that remain connected to at least one server after the removal of some fixed number of clients. Therefore, we seek to count the minimum number of clients that retain a service path after the removal of some fixed number of other clients. Generally, the serviceability of a client can depend on the serviceability of every other client. However, in real world networks, this is rarely the case. We begin by describing how to isolate the serviceability of a subgraph in an arbitrary client-server graph.

For arbitrary $G = (S \cup C, E)$ define the set of *gateways*, $Y \subseteq C$, as those clients that are adjacent to at least one server in S. The remaining clients are labeled *children* are placed in the child set L. A subset of children, L', are said to be *siblings* when they form a single connected component after all gateways are removed. For each set of siblings L', there is a corresponding subset of gateways, Y' that are adjacent to the children in that set. We call such a subset Y' the gateways of siblings L' and conversely, we call siblings L' the children of gateways Y'. Together, the set $Y' \cup L'$ is called a *family*. The serviceability of the children in a family depends entirely on their connectivity to the gateways of that family.

OBSERVATION: If F is a family in G and c is a child in F, then c is serviceable iff there is a path from c to some gateway in F.

According to the above observation, we can decouple the serviceability of clients in different families. This leaves us with a divide-and-conquer strategy for solving $MinConn(G, n, 0)$ as Knapsack Problem.

PROCEDURE: $MinConnDC(G, n, 0)$

Let G be a client-server graph. Bound the solution to Problem $MinConn(G, n, 0)$ as follows.

- Form an exhaustive and disjoint decomposition of G into families $\mathcal{F} = F_1, F_2, \ldots$

- Form a knapsack subproblem as follows. To each family $F_i = (Y_i, L_i)$, assign a maximum weight $W_i = |Y_i|$ and unit value multiplier U_i so that for any quantity, x, of removed clients in F_i, the value xU_i bounds from below the number of clients that *are not* serviceable.
- Solve the knapsack problem for maximum aggregate weight n where item i corresponds to family F_i, with weight $x \in [0, W_i]$ and value xU_i. The optimal value is a lower bound on client connectivity. The gateway clients associated with items chosen can be removed from G to find a corresponding upper bound.

The remainder of this section is devoted to providing details on the execution of each step in this procedure. The variety of knapsack problem that we use can be solved by greedy choice as it is similar to the Fractional Knapsack Problem. To do so, we begin by sorting items according to decreasing unit value. Items with highest unit value are chosen multiply (item i has multiplicity W_i) until the maximum aggregate weight limit has been reached.

Our next step is to derive the maximum weight and unit value multiplier functions for an arbitrary client-server graph G. The first step is to decompose G into exhaustive and disjoint families. We begin by identifying all families in G, then independently measure how their serviceability is affected by client removal, and finally combine the results for each family. Algorithm 1 addresses the first task for client-server graph $G = (S \cup C, E)$. In this algorithm, subroutine NEIGHBORS(G, c) returns the set of vertices in G that share an edge in E with vertex c. Subroutine COMPONENTS(G) returns a set of sets of vertices, with each set of vertices corresponding to a connected component of G. Subroutine CLOSURE$(\mathcal{F})$ returns the transitive closure of the union operation on each set $F \in \mathcal{F}$ — i.e., subroutine CLOSURE$(\mathcal{F})$ combines overlapping sets in $\mathcal{F}$ until all families in $\mathcal{F}$ are pair-wise disjoint.

Algorithm 1 FAMILIES(G)

```
1:  Y ← ∅                                                  (gateways)
2:  for s ∈ S do
3:      Y ← Y ∪ NEIGHBORS(G, s)
4:  end for
5:  L ← C \ W                                              (children)
6:  B ← COMPONENTS(G(L))                                   (siblings)
7:  F ← ∅                                                  (families)
8:  for B ∈ B do
9:      F ← ∅
10:     for b ∈ B do
11:         F ← F ∪ NEIGHBORS(G(L ∪ Y), b) ∪ {b}
12:     end for
13:     F ← F ∪ {F}
14: end for
15: return CLOSURE(F)
```

Lines 1–4 are devoted to delineating two sets Y and L, which contain all gateways and children respectively. In line 6, families of G are initially formed as the connected components of the subgraph of G induced by the set of all children L. In lines 8–14, each family is enlarged to include the gateways for each set of siblings. Finally, in line 15, we combine families that share common gateways so that each family is disjoint from every other family.

Algorithm 2 MULTIPLIER(Y, L)

```
 1: if L ≡ ∅ or |Y| ≡ 1 then
 2:    return |Y ∪ L| / |Y|
 3: end if
 4: x ← |Y|, y ← |L|, U ← y/x          (best initial guess)
 5: P ← POWER(Y)
 6: for P ∈ P do
 7:    for i = 1 to |Y| − |P| do
 8:       r ← MinnConnSDP(G(Y ∪ L) ⊖ P, i, 0)
 9:       y* ← y − r                   (clients out of service)
10:       x* ← i + |P|                 (clients removed)
11:       if x* ∗ U > y* then
12:          U ← y*/x*                 (choose highest ratio)
13:       end if
14:    end for
15: end for
16: return U
```

Let $\mathcal{F}$ be the set of all families in G as constructed by Algorithm 1. We next seek to characterize the serviceability of each family $F \in \mathcal{F}$ by a pair of numbers: the maximum weight and the unit value multiplier, which we label $W(F)$ and $U(F)$ respectively. The maximum weight will be equal to the number of clients that must be removed in order to *completely* disconnect F from every server, and the unit value multiplier will be formulated so that $x\ U(F)$ bounds from above the number of *non-serviceable* clients in F after the removal of any x clients. The maximum weight of family F is always equal to the number of gateways in F because on the one hand, there is no way to indirectly disconnect a gateway from all servers (which bounds $W(F)$ from below), and on the other hand, every child in F must ultimately contain at least one gateway on every service path (which bounds $W(F)$ from above). Constructing the unit value multiplier is more difficult because there are many ways to remove clients in F and we must take care to find the most damaging combination *for each* quantity on the interval $[1, \ldots, W(F)]$. Algorithm 2 shows how to construct $U(F)$ for an arbitrary F and according to our requirement that $U(F)$ yield an upper bound on non-serviceability. The subroutine POWER(Y) returns the power set of set Y, or a set of all possible subsets of Y.

Algorithm 2 begins by testing for trivial cases where we can immediately identify the unit value multiplier. For all other cases (line 4), we begin by setting the

multiplier to the most obvious ratio where all gateways are removed. It is possible that a better solution exists, and by iterating on all possible subproblems of this family, we progressively refine this choice of unit value multiplier. Line 8 finds the solution to each subproblem by running it through $MinConnSDP(G, n, 0)$. Note that in Section 2, we worked with relatively large graphs that did not immediately yield a strong solution to the semidefinite program, and therefore required Procedure *MinConnBC* for strengthening the result. For the subproblems in Algorithm 2, however, we find that the problems are small enough that further analysis is not necessary. The solution to $MinConnSDP(G, n, 0)$ gives a *lower* bound on the number of clients that remain serviceable after removing some n clients. For y^*, however, we require an *upper* bound on the total number of clients that are removed from service after removing n clients. Line 9 reverses the output from *MinConnSDP* from number of clients connected to number disconnected. The remaining lines replace the old multiplier value only if the new value is greater.

3.1.1 Managing Subproblem Size

Algorithm 2 iterates over the power set of Y. This set becomes quite large for even small sets, so we aim to keep the number of gateways at 5 or fewer. However, we continue to require that families remain pair-wise disjoint so we must find a way to *split* families with large gateway sets. For the purpose of creating a lower bound on $MinConn(G, n, 0)$, it's clear that removing edges will only lower the value of the solution and will not jeopardize the bound. So, we judiciously choose edges to remove so that there are 5 or fewer gateways corresponding to each family in the new graph. Since we are dealing with relatively small subgraphs, a simple spectral approach will suffice.

Ding et. al [14] showed how to use the eigenvector corresponding to the second smallest eigenvalue of the Laplacian Matrix of a graph to find nearly-disconnected components of that graph. Consider the graph $H = G(F)$ for any family F. The spectrum of the Laplacian of H has one eigenvalue equal to 0 for each connected component in H. In this case, zero-valued eigenvalues are the smallest because the Laplacian is always real and symmetric so its eigenvalues are always non-negative. Any eigenvector belonging to a 0 eigenvalue will have the same value for all indices that correspond to vertices in the same connected component. Matrix perturbation theory predicts that graphs that have nearly disconnected components will show similar eigenvector values between indices corresponding to vertices in each component when considering the eigenvector corresponding to the second smallest eigenvalue. The following procedure outlines the steps that we took to to break families with large gateway sets into groups of families with no more than 5 gateways.

Let $F = (Y, L)$ be a family in client-server graph G and define subroutine EIGENVECTOR(H) to be a function that returns the eigenvector corresponding to the second smallest eigenvalue of the Laplacian of H. If $|Y| > 5$, then it can be disassembled with the following steps.

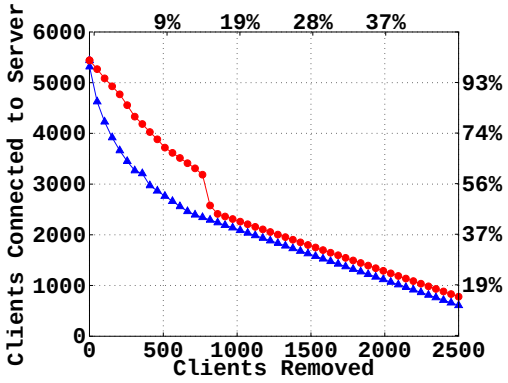


Figure 6: AS graph connectivity

- Compute $x \leftarrow \text{EIGENVECTOR}(G(F))$
- Let $k = \lceil |Y|/5 \rceil$, and break Y into k subsets labeled $Y_1, \ldots, Y_k$ where $|Y_i| = 5$ for $i < k$ and $|Y_k| = |Y| - 5(k-1)$.
- Assign clients of Y to one of the subsets so that clients in the same subset have similar entries in eigenvector x.
- Create k new families $F_1, \ldots, F_k$ so that family F_i has set Y_i as its gateways, and create child sets L_i for each family F_i with L_i initially empty.
- Assign each child in L to the family F_i whose average eigenvector entry matches closest to its own eigenvector entry.

3.2 Evaluation on the Internet AS Graph

Our first evaluation is on a snapshot of the Internet at the Autonomous Systems (AS) level taken by the CAIDA project in April 2005 [13]. Their taxonomy identifies *inferred* Tier-1 and Tier-2 ASes. We create a client-server graph with Tier-1 and Tier-2 ASes labeled as servers and clients, respectively, and with an edge set derived from the corresponding list of peering relationships. We assume that the backbone of the Internet (the Tier-1 ASes) remains fully intact, but that Tier-2 ASes are vulnerable to removal due to attack or administrative-level disenfranchisement. We don't claim that this is the most likely attack on the Internet; it is rather presented as a means of understanding how node removal at the *edge* of the Internet influences overall connectivity.

Let G denote the client-server graph that models the AS graph. We use *MinConnDC* to compute upper and lower bounds on $\mathit{MinConn}(G, n, 0)$ for various values of n. Figure 6 shows the results where the number of clients removed (n) varies from 0 to 2,500 in increments of 50. The blue curve in triangles is the lower bound delivered as the solution to our knapsack problem. The red curve in circles is an upper bound developed by actually removing from G those client vertices identified by the knapsack solution.

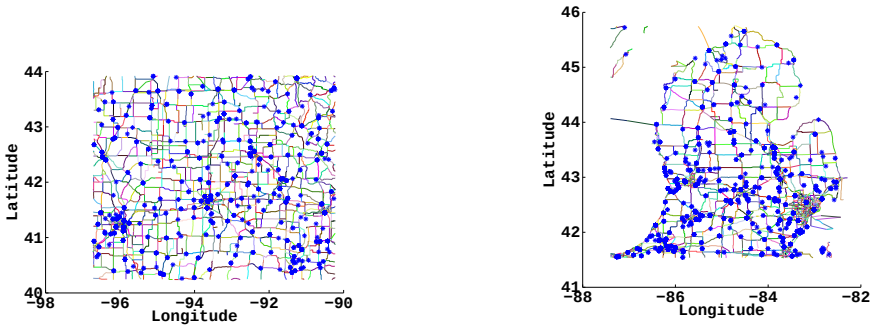
In contrast to the performance of $MinConnBC(G, n, 0)$ from Section 2, the Knapsack solution has somewhat weak performance for small values of n, but tightens considerably for larger values. The initial weaker performance can be attributed to the error introduced by splitting families with large gateway sets. Algorithm 1 returned 2,759 families total, and among those there were 9 that had more than 5 gateways vertices. The largest set was found in one family that had 292 gateways. The rest had fewer than 30 gateways. By splitting the largest gateway set almost 60 times we introduced error. The lower bound makes it seem *easier* to disconnect service paths to a larger number of clients than is actually possible. However, once n is large enough that this large gateway set can actually be removed, the upper bound settles close to the lower. We believe that this type of tradeoff will be typical when applying *MinConnDC* to any client-server graph with such hierarchical structure. Accuracy will largely depend on the extent to which the graph can be decomposed into families with the small gateway sets.

In our analysis we identified a small handful of families responsible for almost all of the complexity in the graph. They had large gateways sets and large sets of children. The remaining families had small gateway sets and small sets of children. They appear as commodities to the knapsack algorithm. Overall, the AS graph exhibits strong serviceability in the face of significant deterioration of Tier-2 clients. Even when 900 ($> 16\%$) Tier-2 clients are removed, better than half the Internet remains in service. This result appears in sharp contrast to research outlined in Section 1.3 where both Tier 1 and Tier 2 nodes can be removed. There, far fewer than 50% of all nodes remain in the connected component even when only 3% of the nodes have been removed [21, 33].

3.3 Evaluation on Airport Connectivity

We next considered airport connectivity in the states of Iowa and Michigan. To do so, we began by obtaining U.S. highway information from the Oak Ridge National Laboratory [2]. These were found in a single large file composed of more than 200K different routes in North America. Each route was delineated by a sequence of coordinates. We call these coordinates *waypoints*. From the Socioeconomic Data and Applications Center (SEDAC) [1], we obtained population density estimates from the 2000 census that are accurate to within several hundred meters. We chose airports as only an example of critical infrastructure, and as in our above cases, there are limitations to our analysis. We make the simplifying assumption that each airport is equally resourceful. We also do not claim this is the most effective attack on infrastructure, but it does server to bound the severity of a particular kind of coordinated attack on a very import piece of national infrastructure.

For each state, we extracted the routes and corresponding waypoints that fall within the bounding box containing that state. A graph representing the highway connectivity of each state was then generated by creating a client vertex for each waypoint and an edge for any two adjacent waypoints in a given route. Because one route will often end at the beginning of another, we consolidated



(a) Iowa (b) Michigan

Figure 7: Airports are shown as blue dots over highways in (a) Iowa and (b) Michigan. Colors vary between different routes. Waypoints are not shown.

waypoints that were within 100 meters of each other into a single vertex. Our next step was to create a server vertex for each airport, which we linked to all waypoints within a (roughly) six-mile radius. The coordinates for airports in these states were found at the data warehouse site socrata.com [3]. The final step in graph construction was to assign a value to each waypoint corresponding to the estimated population density near that point using the SEDAC data. Since people might arrive at an initial waypoint near their place of residence by means other than automobile, we averaged the measured population density between all waypoints within approximately 15 miles of each other and assigned each waypoint a value corresponding to this population density. This means that the sum of all waypoint values in a given state is approximately equal to the population of that state. Figure 7 shows all the routes used for each state as well as the available airports.

Our construction yielded a client-server graph with clients corresponding to waypoints in the given state and whose value was approximately equal to the portion of the state's population residing near the waypoint. Our objective was to bound the quantity of people who maintain highway access to *any* airport after the removal of some fixed number of waypoints. We imagine that the waypoints are removed by either natural disaster of coordinated attack. We constructed a solution to the *WeightedMinConn* Problem for this scenario by modifying *MinConnDC* as follows: *i)* client values were incorporated by first modifying Algorithm MULTIPLIER to return the highest ratio of client *values* to gateways removed, *ii)* the objective function of Program *MinConnSDP* was altered to minimize the aggregate value of connected clients.

3.3.1 Iowa

To apply our *WeightedMinConn* solution, we constructed a client-server graph corresponding to the state of Iowa. The boundary of the state was delineated by

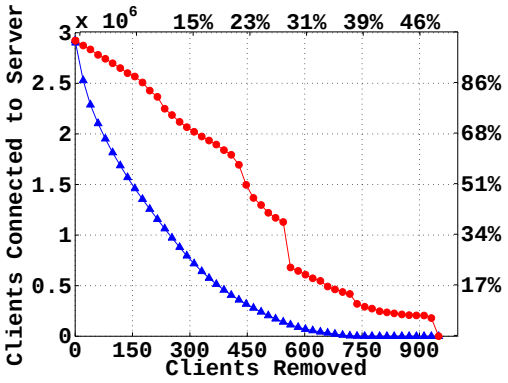


Figure 8: Airport connectivity in Iowa

the coordinates (-104.3701,42.6738) and (-96.2842,46.1566). In total there were 1947 waypoints and 272 airports. Algorithm FAMILIES returned 493 families total with just three families having more than 5 gateways. They had gateway sizes 8, 19, and 501. Breaking the family with 501 gateways into 101 subfamilies was the main source of error. It causes the lower bound to retreat substantially from the upper for small numbers of removed waypoints.

Figure 8 shows that despite the somewhat loose agreement between upper and lower bounds, we can still draw interesting conclusions about how airport access degrades with waypoint removal. For example, when just 200 waypoints (approximately 10%) are blocked, it is possible to disconnect at least 500K (15%) people from every airport, but no more than 1.7M (50%) can possibly be disconnected. There is also a sudden drop in the upper bound after approximately 575 waypoints are removed. This is the point when the largest family (the one that began with 501 gateways) is finally completely disconnected. It's a critical point where the number of people connected drops by about 500K.

3.3.2 Michigan

We performed similar analysis for the state of Michigan where the corresponding client-server graph contained 2207 waypoints and 285 airports. We chose to focus on the lower peninsula of Michigan (the mitten) bounded by the coordinates (-87.4072,41.5327) and (-82.2656,45.7593). After applying Algorithm FAMILIES there were 806 families total with 15 having more than 5 gateway vertices. The three largest gateway sets were 129, 112, and 53. All other families had fewer than 25 gateways. Figure 9 shows how airport connectivity degraded with waypoint removal. Again, the families with large gateway sets were the major source of error in the lower bound. They forced a large gap between upper and lower bounds for up to 300 removed waypoints. Subsequently, the bounds tightened significantly.

From the plot we can see that no more than 4.3 million people (43%) will

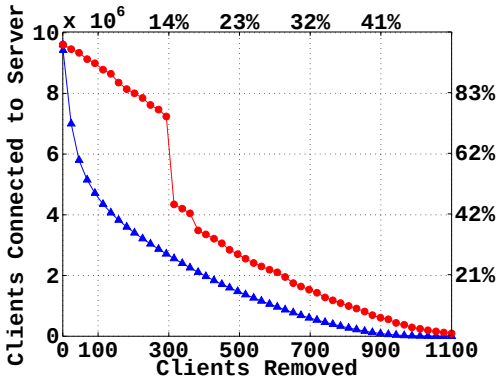


Figure 9: Airport connectivity in Michigan

maintain airport connectivity after 300 waypoints have been removed, and it's possible that as few as 2.8 million (28%) will maintain connectivity. In other words, at least half the residents of Michigan will lose airport access if fewer than 14% of the waypoints are removed.

4 Computational Complexity

Wolcowicz and Zhao studied a problem fundamentally similar to *MinConn*: Graph Partitioning. They sought to discover the fewest number of edges whose removal could partition a graph into disconnected blocks with sizes $n_1, \ldots, n_b$. Their idea was to first assign an indicator vector to each block $x^1, \ldots, x^b$, where $x^i_j = 1$ iff vertex j is assigned to block i. The ideal optimization algorithm would assign the x^i so that $|x^i| = n_i$ and the number of edges between vertices in different blocks was minimized. Counting the number of edges between blocks can be accomplished follows: label edges passing between blocks as *external*. These edges compose the *edge cut*; its cardinality is called the *edge cut size*. It can be verified that $L \bullet x^i(x^i)^T$ gives the number of external edges incident to vertices in block i [9]. It follows that $\frac{1}{2}\sum_i L \bullet x^i(x^i)^T$ gives the edge cut size. A simpler version of this problem can be realized for both edge and vertex partitions on two blocks as follows.

PROBLEM: $EdgePartition(H, n_1, n_2)$

- **Given:** Graph $H = (V, E)$ and desired blocks sizes n_1 and n_2, with $n_1 + n_2 = |V|$.
- Find: minimum quantity of edges whose removal creates two blocks with sizes n_1 and n_2.

PROBLEM: $VertexPartition(H, n_1, n_2)$

- **Given:** Graph $H = (V, E)$ and desired blocks sizes n_1 and n_2, with $n_1 + n_2 = |V|$.
- Find: minimum quantity of vertices whose removal creates two blocks with sizes n_1 and n_2.

Bui and Jones [11] showed that weaker versions of these problems are NP-hard. The version they analyzed looked for the smallest cut that separates a graph into two blocks, each no smaller than $\alpha|V|$, with $\alpha \leq \frac{1}{2}$. They showed that it is NP-hard to find a bipartition less than $\text{OPT} + n^{2-\epsilon}$ for edge cuts and less than $\text{OPT} + n^{0.5-\epsilon}$ for vertex cuts where $\epsilon > 0$ in both cases. This implies that Problems EdgePartition and VertexPartition are also NP-hard. To show that problems $MinConn(\Theta, v, 0)$ and $MinConn(\Theta, 0, e)$ are also NP-hard, it will suffice to reduce them to the later.

THEOREM: $EdgeSeparator(H, n_1, n_2) \leq MinConn(G, 0, m)$.

PROOF: Consider the decision problem $DEdgeSeparator(H, n_1, n_2, k)$, which returns 1 when there exists an edge separator of size k or less that achieves balance (n_1, n_2). Assume without loss of generality that $n_1 \leq n_2$. We will show that any instance of *DEdgeSeparator* can be efficiently transformed into an instance of the decision problem $DMinConn(G, 0, m, c)$. $DMinConn(G, 0, m, c)$ returns 1 when at least c clients are disconnected from all servers after removing any m edges and returns 0 otherwise.

For any instance $(H, n_1, n_2, k) \in$ *DEdgeSeparator*, with $H = (V_H, E_H)$, fashion a corresponding instance of *DMinConn*: $(G, 0, m, c)$ as follows. For $G = (S_G \cup C_G, E_G)$, let $C_G = V_H$ and initialize $E_G = E_H$. Let $S_G = \{s\}$ be a single vertex, and add to E_G one edge between s and each client in C_G. Finally, set $c = n_1$ and $m = n_1 + k$. Ignore, for the moment, s and all adjacent edges, call this graph G^*. What remains in G^* is a graph isomorphic to H. Therefore, any optimal cut corresponding to a solution to *DEdgeSeparator* in H will also exist in G^*. That is, the smallest cut in H that creates blocks of size n_1 and n_2 in H will also create blocks of size n_1 and n_2 in G^*. So

$$(H, n_1, n_2, k) \in DEdgeSeparator \Leftrightarrow (G^*, 0, k, n_1) \in DMinConn.$$

But any block of size n_1 in G must additionally cut the n_1 edges adjacent to s before being completely disconnected from the other block. The result follows by combining the size, k, of the block cut in G^* with the size, n_1, of this server cut so that $m = n_1 + k$.

□

COROLLARY: $VertexSeparator(G, n_1, n_2) \leq MinConn(\Theta, 0, v)$.

PROOF: The proof of Theorem 2 also applies here if we change edge cuts to vertex cuts and replace each undirected edge of the form $(s, c_i) \in E_\Theta$ with the path $(s, x_i) \sim (x_i, c_i)$ where each x_i is an additional client vertex.

□

5 Conclusion

In this paper we have introduced an intuitive model, the client-server graph, that is capable of describing many real-world networks more richly than its predecessors. We have also introduced the Problem *MinConn*, which attempts to quantify the minimum number of client vertices connected to a server after either edge or client removal. Our framing of, and solution to, this problem is preferable to existing techniques such as localized heuristics and graph separators because it offers concrete bounds for edge, vertex and mixed edge-vertex removal in any graph, and it also differentiates between resource providers (servers) and consumers (clients). Our methods are also easily extended to find the minimum aggregate *weight* of serviceable clients after weighted client removal. Finally, our solution leverages a graph's topology to provide bounds for large graphs, a domain where other techniques often fall short.

We used our solution to *MinConn* to demonstrate how client serviceability degrades in various real-world networks. This analysis enabled us to see that the sparse connectivity of the Philippine Power Grid makes it much more susceptible to link attack than the relatively dense Net Equality Wireless Mesh Network. We also saw that the Internet AS level graph is reasonably robust to attacks on the edge of the network. Lastly, we showed that coordinated attacks on the highway systems in Iowa and Michigan could impose large-scale limitations on personal mobility.

References

[1] http://sedac.ciesin.columbia.edu.

[2] http://www-cta.ornl.gov/transnet/highways.html.

[3] http://www.socrata.com.

[4] www.haciendacdc.org.

[5] www.transco.ph/gridmap/2006

[6] M. Amin. Security challenges for the electricity infrastructure. *Computer*, 2002.

[7] M. Amin. North america's electricity infrastructure. *IEEE Symposium on Security and Privacy*, 2003.

[8] S. Aref and B. Miller. Damage information and reporting tool. Technical report, Common Ground Alliance, 2005.

[9] E. R. Barnes and A. J. Hoffman. Partitioning, spectra and linear programming. *Progress in Combinatorial Optimization*, 1984.

[10] C. A. Berrick, S. Morris, and G. M. Malavenda. Federal highway report. Technical report, USGAO, 2009.

[11] T. Bui and C. Jones. Finding good approximate vertex and edge partitions in NP-hard. *Inf. Proc. Letters*, 1992.

[12] R. Cohen, K. Erez, D. ben Avraham, and S. Havlin. Breakdown of the Internet under intentional attack. *Phys. Rev. Let.*, 2001.

[13] X. Dimitropoulos, D. Krioukov, G. Riley, and K. Claffy. Revealing the autonomous system taxonomy. *Passive and Active Measurement Conference*, 2006.

[14] C. Ding, X. He, and H. Zha. A spectral method to separate disconnected and nearly-disconnected Web graph components. *Conf. on Knowl. Disc. and Data Mining*, 2001.

[15] A. Flaxman, A. Frieze, and J. Vera. Adversarial deletion in a scale free random graph process. *Sym. on Disc. Alg.*, 2005.

[16] S. Gorman, S. L., R. Kulkarni, and R. Stough. The revenge of distance: Vulnerability analysis of critical information infrastructure. *Jrnl of Cont. and Crisis Mgmt.*, To appear.

[17] C. Helmberg. Semidefinite programming. *European Journal of Operational Research*, 1999.

[18] A. Hickson. Terrorist threat to u.s. highway system. Technical report, U.S. DHS, 2006.

[19] S. Kent, C. Lynn, J. Mikkelson, and K. Seo. Secure Border Gateway Protocol (S-BGP) — Real World Performance and Deployment Issues. In *NDSS*, February 2000.

[20] Y. W. Law, L. van Hoesel, J. Doumen, P. Hartel, and P. Havinga. Energy-efficient link-layer jamming attacks against wireless sensor network MAC protocols. In *SASN*, Nov 2005.

[21] D. Magoni. Tearing down the internet. *IEEE Journal on Selected Areas in Communications*, 2003.

[22] S. Murphy. BGP Security Vulnerabilities Analysis. Technical report, IETF draft, February 2002.

[23] A. Perrig, J. Stankovic, and D. Wagner. Security in wireless sensor networks. *Wireless sensor networks*, 2004.

[24] K. Poulsen. The backhoe: A real cyberthreat. *Wired*, Jan 2006.

[25] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. In *SIGCOMM*, 2006.

[26] A. L. Rosenberg and L. S. Heath. *Graph Separators with Applications*. Springer, 2001.

[27] B. Smith and J. Garcia. Securing the Border Gateway Routing Protocol. In *Proc. of Global Internet*, 1996.

[28] C. W. Taylor. The anatomy of a power grid blockout. *IEEE Power and Energy Magazine*, 2006.

[29] D. Watts. Security and vulnerability in electric power systems. *35th North American Power Symposium*, 2003.

[30] H. Wolkowicz and Q. Zhao. Semidefinite programming relaxations for the graph partitioning problem. *Discrete Applied Mathematics*, 1996.

[31] W. Xu, W. Trappe, Y. Zhang, and T. Wood. The Feasibility of Launching and Detecting Jamming Attacks in Wireless Networks. In *Proc. ACM Mobihoc*, pages 46–57, May 2005.

[32] H. Yang, H. Y. Luo, F. Ye, S. W. Lu, and L. Zhang. Security in mobile ad hoc networks: Challenges and solutions. *Topics in Wireless Security*, 2004.

[33] S. Zhou and R. J. Mondragon. Redundancy and robustness of AS-level Internet topology and its models. *Electronics Letters*, 2004.