Diffusion on complex networks: algorithmic foundations

Anil Kumar S. Vullikanti
Dept. of Computer Science and Virginia Bioinformatics Institute
Virginia Tech
Acknowledgements

Joint work with:

- NDSSL members including: Madhav Marathe, Henning Mortveit, Maleq Khan, Zhao Zhao, Chris Kuhlman, Samarth Swarup
- S. S. Ravi, SUNY Albany
- Dan Rosenkrantz, SUNY Albany
- Aravind Srinivasan, University of Maryland
- Rajmohan Rajaraman, Northeastern University
- Ravi Sundaram, Northeastern University
Diffusion models and applications

- Graph connecting different entities
- Nodes change state based on their neighbors’ states

- Fads and Social conventions
- Adoption of technology
- Cascading failures
- Threshold models
- Stochastic Diffusion, percolation
- Chip firing, sandpiles
- Constrained Network flows
- Power grid
- Packet flows

Wide variety of models for different applications
Broad goals

- **Motivation**: wide variety of diffusion processes are used in different applications
  - Can be formulated by similar fundamental questions in terms of graph dynamical systems

- **Goals**:
  - Modeling and analysis of complex networks
  - Characterize dynamical properties, especially in terms of the underlying graph structure
  - Techniques to optimize and control dynamics
Challenges

- Underlying complex networks
  - Analytical approaches based on renormalization and differential equation methods not easily applicable
  - Network structure not well understood - need for better models

- Characterization of dynamical properties
  - Need to identify new properties

- Need for new scalable computational approaches
  - Poor locality
  - High expansion and large separators
  - Dynamics on and of networks
  - Co-evolution between networks and diffusion process
  - Behavioral changes
Models of Complex Networks

- **Erdos-Renyi, Chung-Lu models**
  - Very simple models
  - Main goal: capture degree distribution
  - Well understood analytically

- **Small world, PA, copying models**
  - Based on hypotheses of social evolution ("rich get richer", etc.)

- **HOT model**
  - Combination of optimization objectives with random evolution

- **First principles approaches: synthetic networks**
  - Most realistic
  - Useful for network/policy planners
  - Need lot of data, models and HPC tools

---

[Li, et al., 2005]
[Borner, et al., 2007]

[Increasing complexity and realism diagram]
Beyond degree distributions

- Edge swap operations
  - Preserves degree distribution
  - Polynomial time mixing [Feder et al., 2005]
- Disease dynamics not completely determined by degree distribution
- Need random graph models to preserve non-local properties

Changes in static network properties with edge swaps

Changes in disease dynamics with edge swaps
Outline for the rest of the talk

- Part I: Modeling and analysis of complex networks
  - Map-reduce based algorithms for relational subgraph analysis
- Part II: Dynamical properties: mathematical and computational aspects
  - Characterize different local diffusion models and techniques for controlling dynamical properties
- Part III: Simulation tools for diffusion models
  - Malware spread in large proximity networks
PART I: MODELING AND ANALYSIS OF VERY LARGE GRAPH PROPERTIES
Summary of contributions

“First principles” approach for synthetic social and infrastructure networks
- Integrates a large number of diverse public and commercial data sets
- Stochastic models that capture properties of real networks

Computing properties of very large networks
- Efficient sampling based approaches for computing structural properties
  - Mapreduce/Hadoop based for relational subgraph analysis
- New parallel algorithms for dynamical properties
General goal

$G = (V, E)$: very large graph

$H = (V', E')$: small template/subgraph

**Goal**: find one or more embeddings of labeled subgraph $H$ in $G$

Non-induced embedding:

$f : V' \to V$ such that

$(u, v) \in E' \Rightarrow (f(u), f(v)) \in E$
Motivation and applications: data mining, social networks, Semantic web

- Detecting fraud in financial transactions [Bloedorn et al.]
- Other applications: connection subgraphs [Faloutsos, et al.]

[Brochelor et al.]: query of the form $v_1, v_2, v_3, p$ such that $v_1$ is a faculty member at UMD, $v_2$ is an Italian university, who is a friend of $v_1$ and $v_3$ has commented on paper $p$ by $v_1$
Motivation and applications: systems, networking and software engineering

- Automatic custom instruction set generation by enumerating convex subgraphs [Bonzini et al.]
- Mining call graphs to localize software bugs [Eichiner et al.]
- Other applications: anomaly detection in networks through subgraph analysis [Noble et al.]

[Maxwell et al.] discovering graph grammars corresponding to memory leaks
Motivation and applications: bioinformatics

[Alon et al.] Characterization of protein-protein interaction networks based on differences in counts of trees on 9 nodes

[Alon] network motifs in transcription networks
Variations: subgraph enumeration problems

- Functions on the space of embeddings
  - Existence and counting all occurrences
  - Functions of labels as part of embeddings
  - Approximate embeddings

- Relational queries
  - Involving node and edge labels
  - Specified by graph grammars

- Motifs and most frequent subgraphs
  - Contrast with random graphs with similar properties

- Graphlets
  - Generalization of degree distribution
Summary of results

- SAHad: randomized Hadoop based algorithm for enumerating trees and tree-like subgraphs
  - For given $\epsilon, \delta$: produces $(1 \pm \epsilon)$ approximation with probability $\geq 1 - \delta$
  - Worst case work complexity bound of $O(2^{2k}m \cdot f(\epsilon, \delta))$
  - Scales to graphs with over 500 million edges and templates of size up to 12
  - Color-coding technique for approximate enumeration

- Heterogeneous computing environments
  - Different clusters and Amazon EC2 without any system level optimization
Summary of results (continued)

- Broad class of relational subgraph queries
  - Node and edge labeled subgraphs
  - Extension to tree-width bounded subgraphs (low treewidth = like trees)
  - Can easily compute classes of distributive functions on the space of embeddings
  - Can extend to weighted and approximate matches
  - Systematic approach to handle queries specified by a class of tree grammars (chain grammars)
  - Graphlets and motifs
Prior approach and challenges

- Large literature on frequent subgraph enumeration and variations: Apriori, FSG, ...
  - Maintain candidate matches for subgraphs with k-1 edges, and extend to subgraph with k edges
  - Backtracking/extensive bookkeeping to ensure valid matches
  - Scales to ~100,000 node graphs, not clear how to parallelize
  - No rigorous worst case bounds
- Database techniques: preprocessing based on fixed distribution of queries
- Dynamic programming based on color-coding
  - No prior parallelization
Attempt: divide and conquer

Could identify embeddings of $H_1$ and $H_2$ and put them together?
But … overlaps possible

Need to keep track of extra information to avoid overlaps between embeddings of sub-templates
Color-coding idea

- **Basic idea**: color graph with $k = |H|$ distinct colors and only count colorful embeddings.
- Dynamic programming to count number of colorful embeddings.
- If $G$ is colored uniformly at random, number of embeddings is proportional to#
colorful embeddings:

  $\Pr[\text{given embedding of } H \text{ is colorful}] = \frac{k!}{k^k}$

  $E[\# \text{ colorful embeddings of } H \text{ in } G] = \frac{k!}{k^k} (\# \text{ embeddings})$

[Alon, Yuster, Zwick, 95]
Dynamic programming for paths

- Let $\text{col}(v)$ denote color of node $v$
- Let $C(v, S)$ be the number of embeddings of colorful paths of length $|S|$ using set $S$ of colors, that end at node $v$.

\[ C(v, \{\ell\}) = \begin{cases} 
1 & \text{if } \text{col}(v) = \ell \\
0 & \text{else} 
\end{cases} \]

\[ C(v, S) = \sum_{w:(v,w) \in E} C(w, S - \{\text{col}(v)\}) \]

Goal: compute $\frac{1}{2} \sum_v C(v, \{1, \ldots, k\})$
Dynamic programming for paths

- Let \( \text{col}(v) \) denote color of node \( v \)
- Let \( C(v, S) \) be the number of embeddings of colorful paths of length \( |S| \) using set \( S \) of colors, that end at node \( v \).

\[
C(v, \{\ell\}) = \begin{cases} 
1 & \text{if } \text{col}(v) = \ell \\
0 & \text{else}
\end{cases}
\]

\[
C(v, S) = \sum_{w: (v, w) \in E} C(w, S - \{\text{col}(v)\})
\]

Goal: compute \( \frac{1}{2} \sum_v C(v, \{1, \ldots, k\}) \)

\[
C(1, \bullet \bullet ) = 2 \\
C(3, \bullet \bullet ) = 0 \\
C(4, \bullet \bullet ) = 1 \\
C(2, \bullet \bullet \bullet ) = C(1, \bullet \bullet ) + C(3, \bullet \bullet ) + C(4, \bullet \bullet )
\]
Mapreduce/Hadoop

- Powerful framework for processing large amount of streaming data
  - Developed by Google for web applications [Dean and Ghemavat, 2005]
  - Open source implementation: Hadoop
- Mapreduce internally sorts key-value pairs and reorganizes items with same key value for reducer
  - Mapper produces key-value pairs for each data item
  - Reducer processes all elements with same key
  - Can be repeated multiple times
  - System takes care of producing data streams and sorting
Graph algorithms using Mapreduce/Hadoop

- Finding paths and diameter
- Pagerank and other random walk based measures
- Community detection and clustering problems
- Subgraph analysis
  - Counting #triangles: [Pagh et al., 2011], [Suri et al., 2011]
  - Subgraph enumeration to ~100,000 node graphs: [Liu et al., 2009]
Extension to trees

- Partition $T$ into sub-trees $T = \{T_1, T_2, \ldots, T_r\}$. $T_i$ has children $\text{child}_1(T_i)$ and $\text{child}_2(T_i)$ and root $\rho(T_i)$.

- Recursively compute $C(v, \rho(T_i), S)$: \#colorful embeddings of $T_i$ using set $S$ of colors ($|S| = |T_i|$) in which $v$ is mapped to $\rho(T_i)$ using the following recurrence

$$C(v, \rho(T_i), S) = Z \cdot \sum_{u, s_1, s_2} C(v, \rho(\text{child}_1(T_i)), S_1) \cdot C(u, \rho(\text{child}_2(T_i)), S_2),$$

where the summation is over all valid $S_1, S_2$, $Z$ is a scaling factor.
SAHad: mapreduce implementation

- Partition $T$ into set $\mathcal{T} = \{T_1, \ldots, T_r\}$ of sub-trees. Let $\text{child}_1(T_i)$ and $\text{child}_2(T_i)$ be child sub-trees of $T_i$.
- Color each node in $G$ uniformly at random using colors from $\{1, \ldots, k\}$
- Repeat for $T_i \in \mathcal{T}$
  - **Map**: takes in input of the form $\langle v, \rho(T_j), C(T_j) \rangle$: count for each subset $S \subset \{1, \ldots, k\}$ with $|S| = |T_j|$, neighbors of $v$
    Produces key = $u$, value = $C(T_j)$, for each neighbor $u$ of $v$
  - **Reduce**: takes in input of the form: key = $v$, and values $C(C_1(T_i))$ and $C(C_2(T_i))$
    Use recurrence to compute $C(v, \rho(T_i), S) = \sum_{u \in \mathcal{N}(v)} \sum_{S_1, S_2} C(v, \rho(\text{child}_1(T_i)), S_1)C(u, \rho(\text{child}_2(T_i)), S_2)$, where summation is over all $S_1 \cup S_2 = S$, $|S_1| = |\text{child}_1(T_1)|$ and $|S_2| = |\text{child}_2(S_2)|$
Performance analysis

**Lemma**

Consider any node $v$ and template $T_i$.

- The input and output sizes for a Counter.Mapper instance, corresponding to $v$ and $T_i$ are $O\left(\binom{k}{|\text{child}_1(T_i)|} + \binom{k}{|\text{child}_2(T_i)|} + d(v)\right)$ and $O\left(\left(\binom{k}{|\text{child}_1(T_i)|} + \binom{k}{|\text{child}_2(T_i)|}\right)d(v)\right)$, respectively.

- The size of the input to any CounterReducer is $O\left(\left(\binom{k}{|\text{child}_1(T_i)|} + \binom{k}{|\text{child}_2(T_i)|}\right)d(v)\right)$, and the corresponding work complexity is $O\left(\left(\binom{k}{|\text{child}_1(T_i)|}\right)\left(\binom{k}{|\text{child}_2(T_i)|}\right)d(v)\right)$.

**Theorem**

For any given $\epsilon, \delta > 0$, SAHad produces a $(1 \pm \epsilon)$ approximation to the total number of embeddings with probability at least $1 - \delta$, and has a total work complexity of $O(|E| 2^{2k} e^k \log \left(1/\delta\right) \frac{1}{\epsilon^2})$.
Experiments: setting

<table>
<thead>
<tr>
<th>Network</th>
<th>No. of Nodes (in million)</th>
<th>No. of Edges (in million)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miami</td>
<td>2.1</td>
<td>105.4</td>
</tr>
<tr>
<td>Chicago</td>
<td>9.0</td>
<td>537.9</td>
</tr>
<tr>
<td>GNP100</td>
<td>0.1</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Different templates

Computing resources

- Athena: 42 node cluster with 32 cores per node
- Amazon EC2
Performance analysis: time & space

Performance with MPI based implementation

Performance in Amazon EC2
Variation with #reducers per node

Total running time vs #reducers per node

Distribution of total running time of reducers

- Athena: one disk per node
- Many reducers: contention for disk
## Summary of experimental results

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Computing resource</th>
<th>Template and network</th>
<th>Key observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approximation error</td>
<td>Athena</td>
<td>U7; GNP</td>
<td>Error below 0.5%</td>
</tr>
<tr>
<td>Impact of number of data nodes</td>
<td>Athena</td>
<td>U10; Miami, GNP</td>
<td>scale from 4 hours to 30 minutes with data nodes from 3 to 13</td>
</tr>
<tr>
<td>Impact of #concurrent reducers</td>
<td>Athena, EC2</td>
<td>U10; Miami</td>
<td>Non-monotone variation in running time</td>
</tr>
<tr>
<td>Impact of #concurrent mappers</td>
<td>Athena, EC2</td>
<td>U10; Miami</td>
<td>Time generally constant</td>
</tr>
<tr>
<td>Unlabeled/labeled template</td>
<td>Athena, EC2</td>
<td>Varying templates 7-12</td>
<td>&lt; 35 minutes</td>
</tr>
<tr>
<td>Graphlets</td>
<td>Athena</td>
<td>U5; Miami, Chicago</td>
<td>&lt; 35 minutes</td>
</tr>
</tbody>
</table>
PART II: DYNAMICS & CONTROL
# Diverse diffusion processes

Graph dynamical system: contact graph, node states, local functions, update order

![Graph dynamical system diagram](image)

**Example: ratcheted threshold-2 model**

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
<th>Example Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percolation and extensions: SI/SIS/SIR/Independent cascades</td>
<td>Each <em>red</em> node infects each <em>blue</em> neighbor independently with some probability</td>
<td>Malware, failures, infections</td>
</tr>
<tr>
<td>Complex contagion: threshold and variants</td>
<td>Each <em>blue</em> node switches to <em>red</em> if at least</td>
<td>Spread of innovations, peer pressure</td>
</tr>
<tr>
<td>Non-monotone multi-threshold models</td>
<td>Thresholds for switching from <em>blue</em> to <em>red</em> and from <em>red</em> to <em>blue</em></td>
<td>More complex social behavior</td>
</tr>
<tr>
<td>Voter models</td>
<td>Each node picks the state of a random neighbor</td>
<td>Spread of ideologies</td>
</tr>
<tr>
<td>Constrained network flows</td>
<td>Flows with node/link capacities and additional constraints on paths</td>
<td>Packet flows, traffic, wireless networks</td>
</tr>
</tbody>
</table>
### Key Questions

<table>
<thead>
<tr>
<th>Understanding Dynamical Properties</th>
<th>Computational aspects</th>
<th>Interventions to control the dynamics</th>
</tr>
</thead>
</table>
| ❑ Existence and characteristics of fixed points  
  ▪ E.g.: average #nodes in state 1 in fixed point  
| ❑ Transient lengths  
| ❑ Stability  
  ▪ How do changes in graph, update order or functions alter dynamics?  
| ❑ Who becomes “infected”?  
| ❑ Impact of network structure | ❑ Computing different dynamical properties  
  ▪ *Reachability*: does the system reach specific configurations of interest?  
  ▪ *Predecessor existence*: what kind of configurations could lead to the current one?  
| ❑ Simulation tools that scale to large systems | ❑ Forcing node states: changing local functions  
  ▪ Freezing selected nodes in a specific state  
    • Malware spread (SIS/SIR): anti-virus patches  
    • Influence spread (threshold): choose sources to seed spread  
    • Voter models: make some nodes stubborn | ❑ Changes in the graph  
  ▪ Add/delete edges to indirectly alter dynamics |
Summary of results

- Analyzing dynamical properties
  - Stability in threshold systems
  - Characterization of limit cycles and fixed points in bi-threshold systems
  - Impact of structural properties: identifying static graph properties

- Efficient algorithms for computing dynamical properties
  - Efficient algorithms and scalable simulation tools for computing dynamical properties

- Control and optimization of the dynamics
  - Influence maximization in voter dynamics
  - Critical sets to control diffusion in SIR (e.g., vaccinations) and threshold models (countering influence)
  - Game theoretical analysis of distributed interventions
Specific results: controlling diffusion in threshold systems

- Goal: choose to remove critical set of at most $k$ nodes/edges so that diffusion starting from set $I$ is minimized.
  - Variations: minimize number of infections, maximize number of uninfected nodes, different initial conditions ($S$ chosen from distribution).
- NP-hard to approximate within factor of $O(n^\delta)$ for any $\delta < 1$, in general.
- Bicriteria-approximation for threshold 1 (simple contagion): choose $k/\epsilon$ nodes to remove, so that number of infections is at most $1/(1 - \epsilon)$ times optimal.
- Threshold more than 1: more complex heuristics that work better than high degree based approaches.
Bicriteria approximation for threshold=1

Flow based algorithm

- Construct flow network with $M = \infty$, $b = k/OPT$
- Output edges in minimum $(s, t)$-cut $(S, \bar{S})$

**Lemma**

Let $(S, \bar{S})$ be the minimum $s, t$-cut. Then, $|(S - \{s\}, \bar{S} - \{t\})| \leq 2k$ and the number of infected nodes, $|S - \{s\}|$ is at most twice the optimal.

1. Suppose $(X, \bar{X})$ is the optimal cut in the original graph. Then, $|(X \cup \{s\}, \bar{X} \cup \{t\})| \leq k + b \cdot OPT$.
2. $|(S, \bar{S})| = |(S - \{s\}, \bar{S} - \{t\})| + (|S| - |I|)b \leq k + b \cdot OPT$, which implies $|(S - \{s\}, \bar{S} - \{t\})| \leq 2k$ and $|S| - |I| \leq 2OPT$. 
Bi-threshold model: limit cycles and fixed points

state of $v = \begin{cases} 
1 & \text{if } \geq k^\uparrow \text{ 1-neighbors} \\
0 & \text{if } < k^\downarrow \text{ 1-neighbors} 
\end{cases}$

Non-monotone dynamics: more realistic model of agent behavior

Example with $k^\uparrow = 1, k^\downarrow = 3$

**Theorem**

Consider a bi-threshold system with thresholds $k^\uparrow, k^\downarrow$.

- If the system is synchronous, all limit cycles are of length at most 2.
- If the system is asynchronous: if $k^\downarrow - k^\uparrow \leq 1$, all limit sets are fixed points. Otherwise, there can exist arbitrarily long limit cycles (even on a tree).
- If the system is asynchronous and the graph is a tree, with $k^\uparrow = 1, k^\downarrow = d(v) + 1$, all limit sets are fixed points.
Proof: fixed points for trees

\[ k^\uparrow = 1, \quad k^\downarrow = d(v) + 1 \implies v \text{ switches from 0 to 1 if it has at least one 1-neighbor, and from 1 to 0 if it has at least one 0-neighbor} \]

Proof by induction

- **Without loss of generality, can assume permutation \( \pi \) updates nodes level by level, with the higher numbered levels first.**
- **Inductive hypothesis:** if \( x \to x' \), then for each node \( v \) other than the root, \( x'_v = x_{p(v)} \), where \( p(v) \) is the parent of \( v \)
- **Base case.** \( v \) is a leaf \( \implies \) following cases
  - \( x_{p(v)} = 1 \): irrespective \( x_v \) either 0 or 1, \( x'_v = 1 \)
  - \( x_{p(v)} = 0 \): irrespective \( x_v \) either 0 or 1, \( x'_v = 0 \)
- **Inductive step:** consider interior node \( v \). For each child \( w \) of \( v \), we have \( x'_w = x_v \), by induction. Therefore, as in the case of leaves, we will have \( x'_v = x_{p(v)} \)
- Again by induction, it follows that after the \( i \)th iteration, all nodes in the first \( i \) levels have the same state as the root.
PART III: EFFICIENT SIMULATION TOOLS
Summary of results: efficient computational tools

- EpiFast: Epidemics on large social-contact networks
- EpiCure: spread of malware in proximity networks
- InterSim: HPC framework based on graph dynamical systems
Malware on hybrid wireless networks

- **Malware: from nuisance to a threat**
- **Challenges and tools needed**
  - Multiple scales: Bluetooth to Internet; self-forming; resistant to regulation
  - Need to model mobility, multi-level network representation to capture interactions between humans and devices, and behavioral changes
  - Modeling and simulation of malware spread: more abstract models and efficient simulation tools that scale to large networks
- **Key questions**
  - Detect and understand characteristics of the spread of new worms
  - Identify vulnerable devices and networks
  - Strategies to control the spread: anti-virus patches, quarantining

"Human mobility and wireless networks could abet the spread of mobile malware" – Jon Kleinberg, Nature 2008
Current approaches: broad spectrum

Compartmental model
- Assume complete mixing population
- Random waypoint mobility

Motivating question: approach that captures worm characteristics reasonably well, but scales to very large graphs
Our approach: EpiCure

- **Malware** modeled as a stochastic diffusion processes.
- High resolution models of synthetic social contact networks, mobility and call behavior
  - First principles based approach, integrates over 14 different public and commercial data sets
  - Detailed model of movement and activities of people in urban regions
  - Can explicitly incorporate behavioral changes in model
- EpiCure: HPC modeling and simulation environment to study mobile malware in large dynamic networks
  - **Generic**: can work with generic malware models and networks (user inputs)
  - **Scalable**: Scales to large realistic *dynamic* networks
  - **Expressive**: Allows one to study a large class of adaptive and non-adaptive responses
Approach and key techniques for scaling

- **Approach**
  - Network-based representation
  - Probabilistic timed transition system (PTTS) motivated by human epidemics
  - Bluetooth specific states abstracted out
  - State reduction by offline traversal
  - Threading based optimization
  - Error less than 5%
  - Scales to millions of devices

- **Highly detailed ns-2 based Bluetooth model**

- **Graphical representation**
  - States: susceptible, incubation, infected, inquiry, timeout, infectious
  - Transition probabilities and time units
  - Legend: infectious, timeout, idle

- **Graphical analysis**
  - % infected over time
  - Comparison of models: Detail (d=0.1), Reduced (d=0.1), Detail (d=0.03), Reduced (d=0.03)
Approach and key techniques for scaling

- **Approach**
  - Network-based representation
  - Probabilistic timed transition system (PTTS) motivated by human epidemics
  - Bluetooth specific states abstracted out
  - State reduction by offline traversal
  - Error less than 5%
  - Scales to millions of devices
## Sample results: simulation setup

<table>
<thead>
<tr>
<th>Factorial experiment design</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network</strong></td>
<td></td>
</tr>
<tr>
<td>Area</td>
<td>Chicago Downtown area (zip 60602)</td>
</tr>
<tr>
<td>Demographics</td>
<td>People in age group of 20 – 50 years</td>
</tr>
<tr>
<td>People (devices); locations</td>
<td>30000; 4400</td>
</tr>
<tr>
<td>Smart device ownership</td>
<td>100% - every individual in the demographic has a smart phone</td>
</tr>
<tr>
<td><strong>Simulation</strong></td>
<td></td>
</tr>
<tr>
<td>Replicates</td>
<td>5</td>
</tr>
<tr>
<td>Duration of Simulation</td>
<td><strong>8 hours</strong> (8 AM to 4 PM), typical work schedule</td>
</tr>
<tr>
<td>Initially infected</td>
<td>1%, 5%, 10%</td>
</tr>
<tr>
<td>Wallclock</td>
<td><strong>Max 2 hours</strong> (lower when responses are implemented)</td>
</tr>
<tr>
<td>Infection seed</td>
<td>8 AM</td>
</tr>
<tr>
<td><strong>Sensitivity analysis</strong></td>
<td></td>
</tr>
<tr>
<td>Malware parameters</td>
<td>Idle time, $p_{to}$</td>
</tr>
<tr>
<td>Network parameters:</td>
<td>Market share ($m$), Location Density ($d$)</td>
</tr>
<tr>
<td><strong>Response mechanisms</strong></td>
<td></td>
</tr>
<tr>
<td>Static</td>
<td>Degree and Betweenness centrality</td>
</tr>
<tr>
<td>Device-based detection</td>
<td>Passive self detection, local and centralized signature dissemination</td>
</tr>
<tr>
<td><strong>Results</strong></td>
<td></td>
</tr>
<tr>
<td>Cumulative infection size</td>
<td>$T(q, x)$: time taken to infect $q$ percent of devices when $x$ is varied</td>
</tr>
</tbody>
</table>
Sample results: mobility matters

Activity-based mobility model

Random Waypoint mobility model
Results: controlling malware spread

Setting: detection by activity monitoring

- System call, power signature or behavior based detection
- Require some number of occurrences before detection: “self detection threshold”
- Use self detection for automatic signature generation: local and centralized signature dissemination

Centralized dissemination is more effective than local
Computational contributions

- **Speed and Parallelization**
  - Sequential EpiCure 300X faster than NS-2
  - Speedups are obtained with <5% loss in accuracy
  - Parallel implementation: Hybrid MPI-threads improves efficiency for multi-core clusters

- **Scale and Complexity**
  - Scales to 3-5 million devices
  - Heterogeneous and realistic spatial networks
  - Time Varying Networks

<table>
<thead>
<tr>
<th></th>
<th>500 Devices</th>
<th>3—5 M devices</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ns-2</strong></td>
<td>45-50 hrs</td>
<td>Cannot Study</td>
</tr>
<tr>
<td><strong>EpiCure</strong></td>
<td>10 minutes (0.1% error, comp. NS-2)</td>
<td>1.5 hrs (&lt;5% error comp. EpiCure v1.0)</td>
</tr>
</tbody>
</table>

**New model reduction and algorithmic techniques needed to scale and parallelize:** EpiCure is the first modeling environment that can represent and study malware over urban scale, time varying and heterogeneous networks
## Summary of computational contributions

<table>
<thead>
<tr>
<th>Factors</th>
<th>Simulation based computational models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>1 location</td>
</tr>
<tr>
<td>Temporal Scale</td>
<td>ms. / µs.</td>
</tr>
<tr>
<td>Spatial Scale</td>
<td>meters</td>
</tr>
<tr>
<td>Network size</td>
<td>500 – 1000 devices</td>
</tr>
<tr>
<td>Within-host Malware Model</td>
<td>Detailed implementation</td>
</tr>
<tr>
<td>Mobility model</td>
<td>Random Waypoint, Random Walk, Random Landmark</td>
</tr>
<tr>
<td>Device interaction network</td>
<td>Based on mobility models</td>
</tr>
<tr>
<td>Detection</td>
<td>Can be implemented</td>
</tr>
<tr>
<td>Control mechanisms</td>
<td>Can be implemented</td>
</tr>
<tr>
<td>Network co-evolution</td>
<td></td>
</tr>
</tbody>
</table>
Summary

- Graph dynamical systems
  - Rich framework to capture a wide variety of diffusion phenomena
  - Challenging algorithmic problems, need new computational tools

- Fundamentally new computational challenges
  - Very large heterogeneous graphs
  - Cannot be easily partitioned
  - Non-uniform communication patterns: difficult to parallelize in conventional models
Thank you