Simulating Large-scale Dynamic Random Graphs in OMNeT++

Kristján Valur Jónsson¹ Ýmir Vigfússon¹ Ólafur Ragnar Helgason²

Reykjavik University Reykjavik, Iceland
School of Computer Science {kristjanvj,ymir}@ru.is
KTH Royal Institute of Technology
Laboratory for Comm. Netw.
Stockholm, Sweden
olafur.helgason@ee.kth.se

OMNeT++ Workshop
March 23, 2012
Introduction and motivation

Networked communications systems are an important research topic.

New paradigms impact our everyday lives in new and sometimes unforeseen ways.

Can expect distributed systems to grow both in number and scale.
  - wireless sensors
  - collaborative sensing and sharing
  - the internet of things
  - ...
The case for simulation

- New protocols and distributed applications require careful evaluation.
- **Notoriously hard to test.**
  - Repeatability
  - Access to testbeds
  - Practical size limitations of testbeds
  - ...

- Simulation is an important tool for analysis

- We can specify a *fine grained model*:
  - Mobility characteristics
  - Full protocol stack
  - Detailed infrastructure
  - ...

- Highly domain specific
The case for random graph based simulation

- We may want to model at a more abstract level using some general but realistic system assumptions.
- Handy tool: Random graph models
  - Simulations of networked systems are based on an underlying network graph
  - Random graph models may be applied in case of a non-deterministic network.

Approach

- Construct a high-level approximation of a networked system by picking an appropriate generation algorithm and parameters.
  - Size/scale
  - Dynamism, e.g. mobility – VANETs, MANETs, pedestrian networks.
  - Comm links: range, capacity, protocol, loss model.
  - Means of establishing links: Graph connectivity, degree distribution.
  - ...
**Random graphs and generation algorithms**

**Definition**

**Random graph.** A *random graph* is a graph $G = (V, E)$ in which vertices and edges are determined by some random process.

**Random graph generators**

The *algorithms* used to construct random graphs.

- **Binomial graphs** – Erdős-Rényi.
- **Small world models** – e.g. Watts-Strogats
- **Scale-free models** – e.g. Barabási-Albert

**Our objective**

*Present an approach and a set of components to enable dynamic generation and maintenance of random graphs at runtime within OMNeT++.*

https://github.com/kristjanvj/DRGSimLib
The simulation toolbox

Components

- **Nodes** – a compound node representing simulated objects
- **Node factory** – manages lifetime of nodes
- **Topology** – manages node relations – the *network graph*

Sample scenario
The node factory – Factory

- Dynamic instantiation and destruction of nodes
- Operates independently of the graph generation

Parameters

nodeType  The class name of a OMNeT++ compound module – the type of node to be manufactured.
generateInterval  The node generation interval.
lifetime  The lifetime of generated nodes.
minLifetime  The minimum lifetime of generated nodes.

Volatile parameters – can plug in OMNeT++ standard random functions in ini file.
Flexible generation/maintenance of random graphs.

Architecture:
- controller
- a plug-in generator instance

Parameters

- **topologyGenerator**: The name of a topology generator class implementing the `IBasicGenerator` interface.
- **updateInterval**: The interval in seconds between updates of the edge structure of the graph.
- **snapshotFile**: The name of a file for storing snapshots of the graph topology for off-line analysis.
Communications network represented by a single data structure

- managed by the controller
- via algorithms implemented in the plug-in generator

Direct message passing (with delay) between nodes

⇒ Minimal message object generation
A sample node

- **Node**: the simulated object
- **NIC** required
The TopologyControlNIC

Parameters

dataRate  The data rate in Kbps.

bitErrorRate  A volatile parameter for the bit error rate.

processingDelay  The processing delay in seconds.

propagationDelay  The propagation delay in seconds.

- The **per-node** counterpart to the topologyControl
- *Registers* the host node with Topology upon instantiation
- *De-registers* upon destruction
- Stores *local neighbourhood view*
Plug-ins for graph generation and management

- The **graph generator** is a *plug-in* instantiated at start of simulation via ini file parameter.
- Derived from a *base class*, implementing **IBasicGenerator interface**.
- Users can **derive their own random graph generator classes**

**IBasicGenerator**

- **addNode** Adds a vertex to the graph data structure and creates edges in accordance with the implemented algorithm.
- **void removeNode** Removes a vertex from the graph. All incident edges are also removed.
- **bool update** Periodic updates of the graph edges to simulate dynamic effects other than node churn.
- **void constructInitialTopology** Called after initialization of the topology manager to create edges between nodes instantiated at time zero.
Binomial graph generation

The binomial graph $g_{n,p}$, also known as an Erdös-Rényi graph.

**Generation algorithm**

Generate $n$ nodes $\in V$. For a node $v \in V$, create an edge $e = (u, v) \in E$ with probability $p$ to each node $u \in V \setminus v$. 
\( g_{n,p} \) insertion of nodes

\[ \{ \text{Upon registration of node } v \} \]
insert \( v \) into \( V \), the vertices collection
\[
\textbf{if} \ |V| = 1 \ \textbf{then} \\
\quad \text{return} \\
\textbf{else if} \ |V| = 2 \ \textbf{then} \\
\quad \text{createLink}(u,v) \\
\textbf{else} \\
\quad \text{select node } u \in V \text{ uniformly at random.} \\
\quad \text{createLink}(u,v) \\
\textbf{for} \ each \ z \in V \setminus u \ \textbf{do} \\
\quad \text{createLink}(v,z) \text{ with probability } p \\
\textbf{end for} \\
\textbf{end if} \]
$g_{n,p}$ graph maintenance

{Executed periodically, $T_\Delta$ is the time units since last update.}

$p_r \leftarrow T_\Delta \cdot p_c$

$p_a \leftarrow T_\Delta \cdot p_c \cdot \frac{|E|}{|V|}$

for each $e \in E$ do

remove $e$ with probability $p_r$

end for

for each $v \in V$ do

do with probability $p_a$: pick a neighbor $u \in V \setminus v$ uniformly at random and add edge $(v, u)$

end for
A small $g_{n,p}$ graph evolution experiment

Average degree distribution

- Note the characteristic *Poisson* shape of the curves
- **Not** scale-free
Barabási-Albert (BA) graph generation

Generation algorithm

Generate a new node \( v \) in the graph \( G = (V, E) \). For the node \( v \), create an edge \( e = (u, v) \) with any peer \( u \in V \setminus v \) with probability
\[
p(u) = \frac{k_u}{\sum_{z \in V} k_z},
\]
where \( k_z \) is the degree of a node \( z \in V \).
A small BA graph evolution experiment

Average degree distribution

- Distinctive power-law shape
- Scale-free?
Power law characteristics of a BA graph

![Graph showing power-law distribution with alpha=2.7 and simulation data points.](image)

- Normalized frequency vs. Node degree
- Power-law alpha = 2.7
- Simulation data
Problematic effects of graph dynamism

- *Dynamism* ⇒ graph connectivity not guaranteed
- Fact of life but *complicates* analysis
- Topology *optionally* guarantees connectivity

(i) Cut vertices
(ii) Cut edges

**Remedy**

1. Search *k*-neighbourhood recursively to determine connectivity.
2. Connect components which have not been confirmed as connected.
Application example: GAP

- The Generic Aggregation Algorithm – GAP\(^1\).
- Designed for *network monitoring applications*
- Continuous monitoring in *dynamic* networks.
- Distributed construction and maintenance of a spanning-tree overlay – the aggregation overlay.

---

In-network computation

Each node provides updates on its own schedule.

Example shows synchronous operation for simplicity.

Recipients of updates compute new contributions in-network.
In-network computation

Each node provides updates on its own schedule. An example shows synchronous operation for simplicity.
In-network computation

Each node provides updates on its own schedule.

Example shows synchronous operation for simplicity.

Recipients of updates compute new contributions in-network.
In-network computation

Each node provides updates on its own schedule. Example shows synchronous operation for simplicity. Recipients of updates compute new contributions in-network.
In-network computation

- Each node provides updates on own schedule
- Example shows synchronous operation for simplicity
- Recipients of updates compute new contributions \textit{in-network}
Constant node population

**Experimental setup**

- **BA graph**, $m_0 = 5$, $m = 3$
- 2500 nodes, static population
- Dynamic link reassignments in window $t = [2h, 4h]$.
- Failure detection latency: $1s + \mathcal{N}(0s, 1s)$.
Link reassignments enabled

Aggregate average $p=0.005$
Aggregate average $p=0.01$
Global average
Link reassignments enabled

Aggregate count p=0.0
Aggregate count p=0.005
Aggregate count p=0.01
Dynamic node population

Experimental setup

- BA graph, $m_0 = 5$, $m = 3$
- 1500 nodes at time zero
- Poisson arrivals with $1/\lambda = 5s.$
- Node lifetime drawn from $\mathcal{N}(\mu, \sigma)$
- Initial node lifetime (zero population) drawn from uniform $[0, \mu]$ distribution.
- Churn interval $t = [2h, 10h]$
- Failure detection latency: $100ms + \mathcal{N}(0s, 100ms)$. 
Aggregate – AVERAGE

Churn initiated

Aggregate average N(8h,4h)
Aggregate average N(6h,4h)
Aggregate average N(4h,4h)

Global observation
Aggregate count $N(8h,4h)$
Aggregate count $N(6h,4h)$
Aggregate count $N(4h,4h)$
Aggregate – COUNT. Slow failure detection.

failure detection: $1s + \mathcal{N}(0s, 1s)$.
Conclusions

- **Motivation**: Allow easy construction of random-graph based simulations in OMNeT++
  - Abstract away from complexities of networks/systems, providing means of evaluating protocols with emphasis on end systems.

- **Toolbox**: Provide a set of components to facilitate simulation based on *random graph models* in OMNeT++
  - Generic node class (example only)
  - Node factory
  - Topology manager with plug-in generators

- **Demonstration application**: GAP dynamic aggregation protocol.

- **Generally applicable** tool for graph-based simulation research
  - May even be used for applications unrelated to physical networks.
  - OMNeT++ as a tool to study social networks?