Simulation Notes

Matthew Skrzypczyk

July 2018

1 Simulation Options

This document outlines various simulation parameters that can be used to test the EGP under various scenario regimes

- $--network-config$: Path the the network configuration file to use for the simulation.
- $--results-path$: Directory to store the results under. This will automatically create a timestamped subdirectory containing the simulation data.
- $--origin-bias$: Probability that the request comes from node A rather than node B.
- $--create-probability$: Probability that a CREATE request is submitted at a particular timestep
- $--min-pairs$: The minimum number of pairs to include in a CREATE request
- $--max-pairs$: The maximum number of pairs that can be requested
- $--tmax-pair$: Maximum amount of time per pair (in seconds) in a request. The total max_time of a request is then $num\_pairs \times tmax\_pair$
- $--request-overlap$: Allow request submissions to overlap, this causes requests to be submitted before previous request's max_time has expired.
- $--request-frequency$: Minimum amount of time (in seconds) between calls to CREATE
- $--num-requests$: Total number of requests to simulate
- $--max-sim-time$: The maximum amount of simulated time (in seconds) to allow the simulation to run for. Default allows the simulation code to decide.

- $--max-wall-time$: The maximum amount of real clock time (in seconds) to allow the simulation to run for. Default allows the simulation to run until is has deemd itself completed. When used in combination with $--max-sim-time$ the simulation will end when either of the specified times has been reached.
- $--enable-pdb$: Turn on PDB pre and post simulation

2 Examples

Here I will describe various simulation scenarios and appropriate command line arguments to the simulation script to simulate them.

2.1 Example 1.

A simulation where:

- Requests have equal probability of being created at either node
- Requests are created with probability 0.5
- Requests strictly contain 2 pairs
- Total number of attempted requests is 100
- Only one request is active at any time
- Config file "network_config.json" is used
- Store results in "results" directory
- Simulation runs for 20 simulated seconds

```
python3 simulation.py --network-config network_config.json --results-path
results --origin-bias 0.5 --create-probability 0.5 --min-pairs 2 --max-pairs 2
--num-requests 100 --max-sim-time 20
```

2.2 Example 2

A simulation where:

- Requests originate from node A with probability 0.2
- Requests are always created when attempted
- Requests contain pairs in the range [2,3]
- Requests are allowed to overlap
- Total number of attempted requests is 1

- Requests occur in a frequency of 0.05 seconds
- Config file "/home/user/network_config.json" is used
- Store results in "results" directory
- Simulation runs for 200 real clock seconds

```
python3 simulation.py --network-config /home/user/network_config.json
--results-path results --origin-bias 0.2 --create-probability 1 --min-pairs 2
--max-pairs 3 --request-overlap --request-frequency 0.05 --num-requests 1
--max-wall-time 200
```

2.3 Example 3

A simulation where:

- Requests originate strictly from node B
- Requests are created with probability 0.8
- Requests contain only 1 pair
- Maximum time for a pair per request is 10 seconds
- Total number of attempted requests of 1000
- Requests are allowed to overlap
- Requests occur in a frequency of 2 seconds
- Config file "network_config.json" is used
- Store results in "/home/user/simulation_results" directory
- Simulation runs for maximum of 300 simulated seconds or 600 real seconds

```
python3 simulation.py --network-config network_config.json --results-path
/home/user/simulation_results --origin-bias 1 --create-probability 0.8
--min-pairs 1 --max-pairs 1 --request-overlap --request-frequency 2
--num-requests 1000 --tmax-pair 10 --max-sim-time 300 --max-wall-time 600
```

2.4 Example 4

A simulation where:

- Requests originate equally from both nodes
- Requests are created with probability 1
- Requests contain pairs in the range [10, 20]

- Maximum time for a pair per request is 100 seconds
- Total number of attempted requests of 1000
- Requests are allowed to overlap
- Requests occur in a frequency of 10 seconds
- Config file "network_config.json" is used
- Store results in "results" directory
- Let the simulation run until it has completed.
- Enable the debugger for inspection of local variables before and after simulation

```
python3 simulation.py --network-config network_config.json --results-path
results --origin-bias 0.5 --create-probability 1 --min-pairs 10
--max-pairs 20 --request-overlap --request-frequency 10 --num-requests 1000
--tmax-pair 10 --enable-pdb
```

3 Data Collection

Simulations will record data into a SQLite database under a timestamped directory within the specified results-path. For the EGP simulations one can interact with the database using the following code:

```
import sqlite3
conn = sqlite3.connect(<path_to_sim_data.db>)
c = conn.cursor()

# Showing all tables constructed
c.execute("SELECT name FROM sqlite_master WHERE type='table'")
for table_name in c.fetchall():
    print(table_name)

# Sample output:
('EGP_Creates_0',)
('EGP_OKs_0',)
('Node_EGP_Attempts_0',)
('Midpoint_EGP_Attempts_0',)

# Showing column number, name, and data type
c.execute("PRAGMA table_info(EGP_Creates_0)")
for column_data in c.fetchall():
    number, name, dtype, _, _, _ = column_data
    print(number, name, dtype)
```

```
# Sample output
0 Timestamp real
1 Node ID integer
2 Create ID integer
3 Create_Time real
4 Max Time real
5 Min Fidelity real
6 Num Pairs integer
7 Other ID integer
8 Priority integer
9 Purpose ID integer
10 Success integer

# Extracting Create's from table EGP_Creates_0
c.execute("SELECT * FROM EGP_Creates_0")
for p in c.fetchall():
    timestamp, nodeID, create_id, create_time, max_time, min_fidelity,
        num_pairs, otherID, priority, purpose_id, succ = p

# Extracting ok's
c.execute("SELECT *  FROM EGP_OKs_0")
for p in c.fetchall():
    timestamp, createID, originID, otherID, MHPSeq, logical_id, goodness,
        t_goodness, t_create, succ = p

# Extracting error's
c.execute("SELECT * FROM EGP_Errors_0")
for p in c.fetchall():
    timestamp, nodeID, error_code, succ = p

# Extracting node attempts
c.execute("SELECT * FROM Node_EGP_Attempts_0")
for p in c.fetchall():
    timestamp, nodeID, succ = p

# Extracting midpoint attempts
c.execute("SELECT * FROM Midpoint_EGP_Attempts_0")
for p in c.fetchall():
    timestamp, outcome, succ = p

# Reconstructing matrices:
c.execute("SELECT * FROM EGP_Qubit_States_0")
for p in c.fetchall():
    timestamp = p[0]
    nodeID = p[1]
```

```
m_data = p[2:34]
# Reconstruct matrix, note the '1j * ' used to include the imaginary component
m = matrix([[m_data[i] + 1j * m_data[i+1]
             for i in range(k, k+8, 2)]
             for k in range(0, len(m_data), 8)])
succ = p[-1]
```