The Network Contributor Rewards Model

Nihar Shah

DoubleZero Foundation

July 17, 2025

Outline

Outline

Running the Model in Practice

- Full model in `network_shapley.py` in repo https://github.com/doublezerofoundation/network-shapley
- Readme, long PDF, and example inputs are in repo
- Production version of the code being developed in Rust in a linked repo
- We'll walk through a minimal working example here

Minimal Working Example

```python
import pandas as pd
from network_shapley import network_shapley

private_links = pd.read_csv("private_links.csv")
devices       = pd.read_csv("devices.csv")
public_links  = pd.read_csv("public_links.csv")
demand1       = pd.read_csv("demand1.csv")

result = network_shapley(
    private_links     = private_links,
    devices           = devices,
    demand            = demand1,
    public_links      = public_links,
    operator_uptime   = 0.98,# optional
    contiguity_bonus  = 5.0, # optional
    demand_multiplier = 1.2  # optional
)

print(result)
```

Required Inputs

- **Private links table**: one row per private link (bidirectional) `(Device1, Device2, Latency, Bandwidth, Uptime, Shared)`
- **Device table**: one row per device `(Device, Edge, Operator)`
- **Public links table**: one row per public link (bidirectional) `(City1, City2, Latency)`
- **Demand matrix**: one row per traffic flow `(Start, End, Receivers, Traffic, Priority, Type, Multicast)`

Optional Inputs

- **Operator uptime**: probability that any given operator stays through epoch
- **Contiguity bonus**: latency penalty applied to hybrid private-public routing, avoided through using contiguous routing
- **Demand multiplier**: scalar to adjust demand up

Minimal Working Example

```
import pandas as pd
from network_shapley import network_shapley

private_links = pd.read_csv("private_links.csv")
devices       = pd.read_csv("devices.csv")
public_links  = pd.read_csv("public_links.csv")
demand1       = pd.read_csv("demand1.csv")

result = network_shapley(
    private_links     = private_links,
    devices           = devices,
    demand            = demand1,
    public_links      = public_links,
    operator_uptime   = 0.98,# optional
    contiguity_bonus  = 5.0, # optional
    demand_multiplier = 1.2  # optional
)

print(result)
```

Output

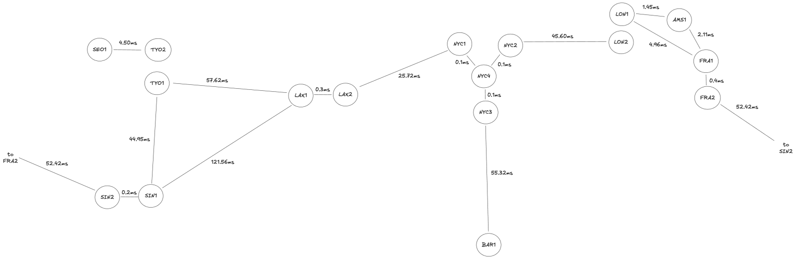
- Returns a `pandas.DataFrame` with columns:

Operator	Name
Value	Raw Shapley value
Percent	$\frac{\text{Value}}{\sum \text{Value}}$

- Links earn rewards based on ability to cut latency and where traffic demand is

- So expect to see large changes depending on where leader is

Simulated Scenario



Simulated Scenario

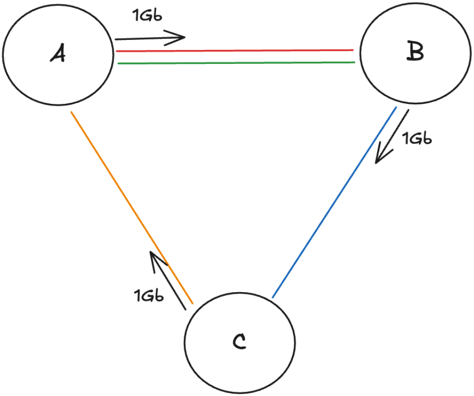
Operator Name	Value #1	Percent #1	Value #2	Percent #2
Alpha	21.5370	2.80%	2.0154	0.16%
Beta	10.6595	1.03%	187.1199	15.01%
Delta	13.5257	1.31%	111.6727	8.96%
Epsilon	0.0407	0.00%	88.5022	7.10%
Gamma	487.1094	47.01%	23.0343	1.85%
Kappa	0.0603	0.01%	10.6422	0.85%
Theta	503.1153	48.55%	333.5523	26.76%
Zeta	0.1393	0.01%	490.0349	39.13%

Outline

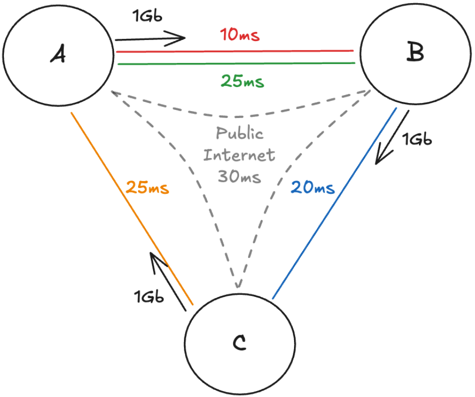
Alternate Idea: Carried Traffic Model

- Carried-traffic model pays only for traffic on each link... seems simple, right?
- But... it is insufficiently discriminating and cannot be kept simple for long
- Shapley values are much more discriminating and robust
- They pay out for the *marginal* contribution to a common **value function**

Illustrative Example



Illustrative Example

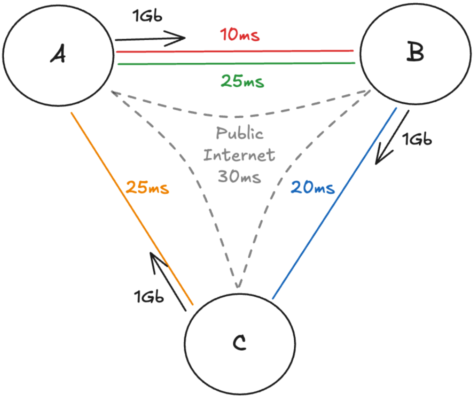


Shapley Values

- Carried-traffic model pays only for traffic on each link... seems simple, right?
- But... it is insufficiently discriminating and cannot be kept simple for long
- Shapley values are much more discriminating and robust
- They pay out for the *marginal* contribution to a common **value function**

Outline

Scenario



Value Function

$$V = -\sum_i t_i l_i$$

- Manifestation of IBRL
- t_i: traffic flow i
- l_i: latency incurred by flow
- Negative sign means we want to maximize this term

Goal: compute this for every coalition of possible contributors

Linear Program Formulation

- To get value for a given set of link operators...

$$\begin{aligned} \text{minimize} \quad & c^\top x \\ \text{subject to} \quad & A_{eq}\, x = b_{eq}, \\ & A_{ub}\, x \leq b_{ub}, \\ & x \geq 0 \end{aligned}$$

- Decision variables x: traffic routed across each directed edge for each traffic type
- Solving yields coalition-specific value $V(C) = -c^\top x^*$

Private and Public Link Inputs

Start	End	Latency (ms)	Operator
A	B	10	Red
A	B	25	Green
B	C	20	Blue
A	C	25	Orange
A	B	30	Public
B	C	30	Public
A	C	30	Public

- Each bidirectional link will be split into two directed edges by the model

Flow-Conservation Matrix

$$\tilde{A}_{eq} = \begin{array}{c} \begin{array}{cccccccccccccc} r & g & b & o & r' & g' & b' & o' & p_1 & p_2 & p_3 & p_1' & p_2' & p_3' \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & -1 & -1 & -1 & 0 & 1 & 1 & 0 & -1 & -1 & 0 & 1 \\ -1 & -1 & 1 & 0 & 1 & 1 & -1 & 0 & -1 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 1 & -1 & 0 & -1 & 1 & 0 & 1 & -1 \end{bmatrix} \end{array}$$

- Rows are nodes and columns are directed edges
- $+1$ if traffic leaves the node, -1 if it enters the node
- Need different flow matrices for different traffic types

$$A_{eq} = \text{diag}\left(\tilde{A}_{eq}, \tilde{A}_{eq}, \tilde{A}_{eq}\right)$$

Demand Vectors

$$\tilde{b}_{eq}^{(1)} = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}, \; \tilde{b}_{eq}^{(2)} = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}, \; \tilde{b}_{eq}^{(3)} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

- Again, need different flow matrices for different traffic types

$$b_{eq} = \begin{bmatrix} \tilde{b}_{eq}^{(1)} \\ \tilde{b}_{eq}^{(2)} \\ \tilde{b}_{eq}^{(3)} \end{bmatrix}$$

Bandwidth Constraints

- Suppose all links but Green have 5 gigabits of capacity; and the Green has 2 gigabits
- Public edges have effectively infinite capacity so there is no constraint imposed on them

$$\tilde{A}_{ub} = [I_{8\times 8} \ 0_{8\times 6}]$$

$$\tilde{b}_{ub} = \begin{bmatrix} 5 & 2 & 5 & 5 & 5 & 2 & 5 & 5 \end{bmatrix}^T$$

- The matrices are stacked horizontally here, because all traffic types draw from the same bandwidth constraint

$$A_{ub} = \begin{bmatrix} \tilde{A}_{ub} \ \tilde{A}_{ub} \ \tilde{A}_{ub} \end{bmatrix}$$

Cost Vector

- Objective minimizes latency, i.e. cost, across traffic types

$$\tilde{c} = \begin{bmatrix} 10 & 25 & 20 & 25 & 10 & 25 & 20 & 25 & 30 & 30 & 30 & 30 & 30 & 30 \end{bmatrix}$$

- Latency is replicated across the three traffic types

$$c = \begin{bmatrix} \tilde{c} \ \tilde{c} \ \tilde{c} \end{bmatrix}$$

- This defines all the components needed for the linear program

Value Function Across Coalitions

Coalition	l_{AB}	l_{BC}	l_{CA}	$V(C)$
No Operators	30	30	30	-90
Red	10	30	30	-70
Green	25	30	30	-85
Blue	30	20	30	-80
Orange	30	30	25	-85
Red, Blue	10	20	30	-60
Red, Orange	10	30	25	-65
Green, Blue	25	20	30	-75
Green, Orange	25	30	25	-80
Blue, Orange	30	20	25	-75
Red, Green	10	30	30	-70
Red, Blue, Orange	10	20	25	-55
Green, Blue, Orange	25	20	25	-70
Red, Green, Blue	10	20	30	-60
Red, Green, Orange	10	30	25	-65
All Operators	10	20	25	-55

Green Link's Marginal Contribution

With G	Without G	ΔV	Weight
$V(G,R,B,O)$	$V(R,B,O)$	$(-55)-(-55)=0$	0.250
$V(G,R,B)$	$V(R,B)$	$(-60)-(-60)=0$	0.083
$V(G,R,O)$	$V(R,O)$	$(-65)-(-65)=0$	0.083
$V(G,B,O)$	$V(B,O)$	$(-70)-(-75)=5$	0.083
$V(G,R)$	$V(R)$	$(-70)-(-70)=0$	0.083
$V(G,O)$	$V(O)$	$(-80)-(-85)=5$	0.083
$V(G,B)$	$V(B)$	$(-75)-(-80)=5$	0.083
$V(G)$	$V(\text{none})$	$(-85)-(-90)=5$	0.250
		Sum	**2.50**

- Summing weighted differences yields Green's Shapley value = 2.5
- Red = 17.5 (50%), Blue = 10 (29%), Orange = 5 (14%), Green = 2.5 (7%).

Outline

Operator Redundancy

- Operators may withdraw for commercial, operational, or regulatory reasons
- This methodology would over-reward fragile topologies
- Solution: pay contributors according to the *expected* value of a network and account for probability p of withdrawals
- This preemptively rewards links with insurance value

$$V = -\mathbb{E}\left(\sum_i t_i l_i\right)$$

Operator Redundancy

- Formally, turn coalition-specific values into expected values:

$$\mathbb{E}\big[V(C)\big] = \sum_{S \subseteq C} p^{|S|}(1-p)^{|C|-|S|}\, V(S)$$

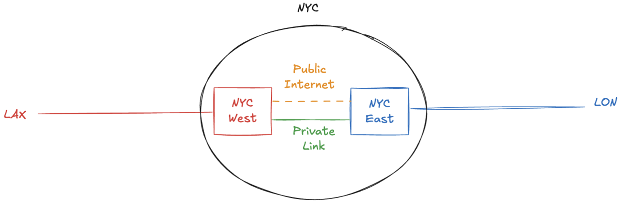
- In practice, this is a computational bottleneck so we use a one-pass algorithm (described in the manual)

$$\mathbb{E}(V) = CBv$$

DZXs, Edges, and Contiguity

- This methodology would ideally incentivize intra-city connections ("DZXs") and network edges too
- But edges are outside the methodology; and the scope for DZXs to improve over the public internet is small given short distances
- Solution is to add a fixed latency penalty to non-contiguous (a.k.a. hybrid) routes, i.e. routes that need to use the public internet to transit through a city or off the network to users
- This encourages end-to-end deployments, but still admits non-contiguous networks as needed

DZXs, Edges, and Contiguity



- Orange public link is charged some penalty
- Contributors only unlock the full improvements for LA-London traffic by building the green private link

Measuring Latency

- Real-world latencies are hard to measure as a single number, because they fluctuate with congestion and size
- Solution: measure latency distributions over sampled intervals and non-trivial flow sizes
- Use an upper percentile (e.g. p95) as the effective latency measurement in the value function
- Brings notion of both speed and jitter into rewards calculus without changing core structure

Scaling Demand

- There may be different priorities of traffic (e.g. stake weight of different receivers); so the value function is extended:

$$V = -\mathbb{E}\left(\sum_i p_i t_i l_i\right)$$

- Also, potentially scale current demand by $\gamma > 1$ to anticipate future growth and expose chokepoints
- Gives incentives to invest in these before demand catches up
- Gradually adjust $\gamma \to 1$ as the network matures

Multicast

- Multicast is the ability for the network, rather than the sender, to duplicate packets
- This conserves bandwidth and allows for higher link utilization
- The methodology already extends to this case, for multicast-enabled demand flows
- All links, except links that move traffic from the network to the public internet, automatically support multicast in DoubleZero

Conclusion

- Check out the repo, readme, and manual
- There is a video of an *earlier* version of this deck and methodology at https://youtu.be/K1Ni-k51sMw
- Contact me at nihar@doublezero.us with questions
- On the horizon...
  1. The primary focus is turning this into production code for launch
  2. As part of that, there is a strong focus on measuring quantities and setting parameters accurately
  3. In the medium term, it would be nice to design a web interface