Fast Network Simulation Setup

Lab 1: AIMS conference 2014

Lorenzo Saino

Communications and Information Systems Group
Department of Electrical and Electronics Engineering
University College London
l.saino@ucl.ac.uk

http://fnss.github.io

Outline

Network experiment scenarios

- Background and motivations
- Topologies
- Traffic matrices

Fast Network Simulation Setup (FNSS) toolchain

- Features
- Architecture
- Download and installation
- Usage

Coding

- Live coding examples
- Coding exercises

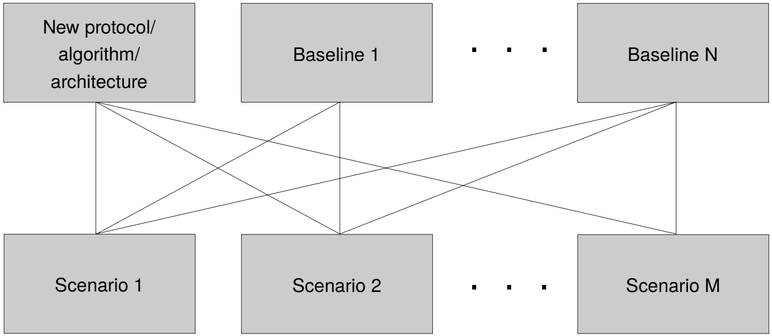
Requirements for coding exercises

Download and install VirtualBox (`http://www.virtualbox.org`)

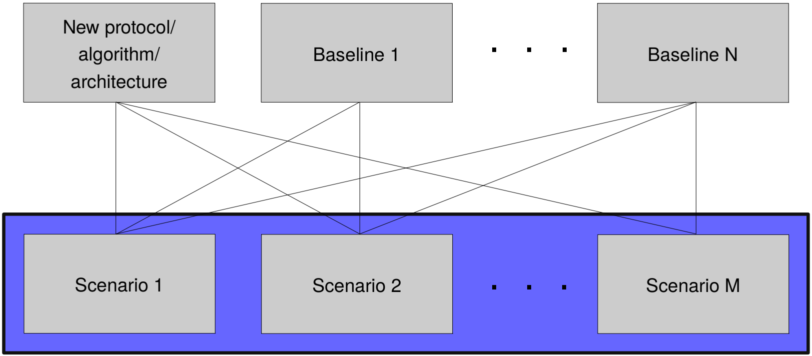
Download FNSS virtual machine image (`http://fnss.github.io`) and start it with VirtualBox

Anatomy of a (controlled) network experiment

Anatomy of a (controlled) network experiment



Anatomy of a (controlled) network experiment



What's a scenario?

What's a scenario?

Scenario = network model + workload

What's a scenario?

Scenario = network model + workload

Network model

- Network topology
- Link characteristics (capacities, delays, weights, ...)
- Node configuration (stacks, applications, buffer sizes, ...)

What's a scenario?

Scenario = network model + workload

Network model

- Network topology
- Link characteristics (capacities, delays, weights, ...)
- Node configuration (stacks, applications, buffer sizes, ...)

Workload

- Traffic matrix
- Application-layer events (e.g., HTTP requests, DNS requests)
- Network events (e.g., node/link failures, node mobility)
- Reconfiguration events

Network topologies

Topologies commonly used for running network experiments generally fall in one of these categories:

Network topologies

Topologies commonly used for running network experiments generally fall in one of these categories:

- **AS-level topologies:** Internet-wide network of Autonomous Systems (AS)

Network topologies

Topologies commonly used for running network experiments generally fall in one of these categories:

- **AS-level topologies:** Internet-wide network of Autonomous Systems (AS)
- **Intradomain topologies:** PoP- or router-level topologies of a specific Autonomous System (AS)

Network topologies

Topologies commonly used for running network experiments generally fall in one of these categories:

- **AS-level topologies:** Internet-wide network of Autonomous Systems (AS)
- **Intradomain topologies:** PoP- or router-level topologies of a specific Autonomous System (AS)
- **Datacenter topologies:** Physical-layer topologies of a datacenter

Network topologies

Topologies commonly used for running network experiments generally fall in one of these categories:

- **AS-level topologies:** Internet-wide network of Autonomous Systems (AS)
- **Intradomain topologies:** PoP- or router-level topologies of a specific Autonomous System (AS)
- **Datacenter topologies:** Physical-layer topologies of a datacenter
- **Simple models:** Simple synthetic topologies

AS-level Internet topology

Business relations

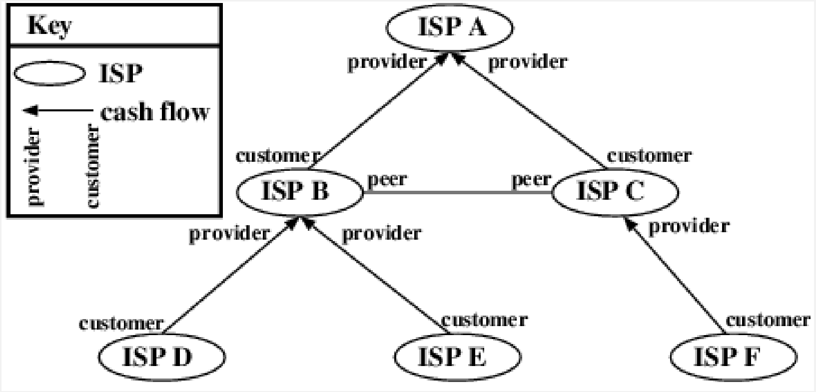


Image courtesy of CAIDA

AS-level Internet topology

Valid and invalid paths

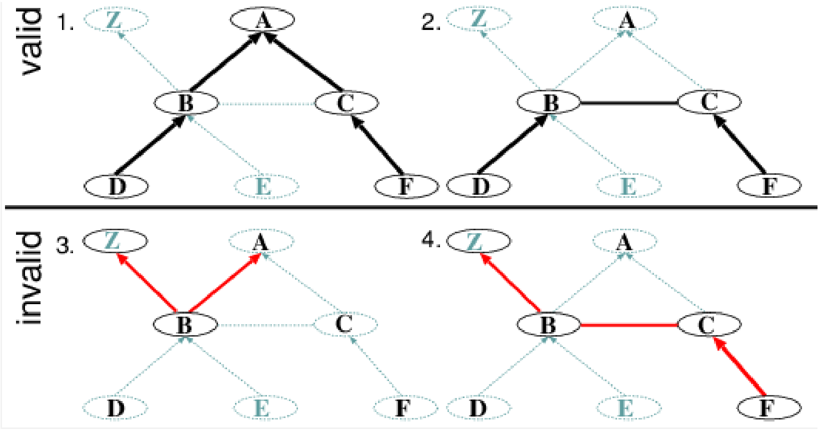


Image courtesy of CAIDA

Inferring AS-level topology

The AS-level Internet topology can be inferred from BGP routing data and Traceroute.

CAIDA maintains a dataset of AS-level Internet topology gathered from publicly-available BGP routing information taken from various vantage points[1].

Known inaccuracies in the CAIDA dataset:

- Inaccurate link type inference
- Missing peering links[2]

[1] `http://www.caida.org/data/active/as-relationships/`

[2] R. Cohen and D. Raz, The internet dark matter - on the missing links in the AS connectivity map, in *INFOCOM'06*

Synthetic models

The AS-level Internet graph is a *scale-free graph*, in which the frequency of nodes having degree k and the degree of the node have a power-law relationship[3]:

$$P(k) \propto k^{-\alpha}$$

This phenomenon is an effect of the *preferential attachment*[4], *i.e.* new nodes joining the network preferentially attach to nodes already well-connected.

[3] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. In *SIGCOMM 99*

[4] A. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509512, 1999

Barabási-Albert model[5]

This model generates random scale-free graphs with:

$$P(k) \propto k^{-3}$$

Formation process:

- Start with a line graphs of m_0 nodes
- Add a new isolated node
- Connect the node to m existing nodes randomly selected according to the following PDF:

$$\Pi(i) = \frac{deg(i)}{sum_{v \in V} deg V}$$

- Repeat until the graph has the desired number of nodes n

[5] A. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509512, 1999

Extended Barabási-Albert model[6]

Overview

This model is an extension of the original Barabási-Albert model which also takes into account some local events such as addition of new links and rewiring.

The output is still a scale-free graph but exponent varies.

[6] R. Albert and A. Barabási. Topology of evolving networks: local events and universality. *Physical review letters*, 85(24):52345237, 2000

Extended Barabási-Albert model

Formation process

Formation process:

- Start creating m_0 isolated nodes
- At each step:
  - With probability p add m links to existing nodes randomly selected according to:

$$\Pi(i) = \frac{deg(i) + 1}{\sum_{v \in V}(deg(v) + 1)}$$

  - With probability q disconnect m links and reconnect them according to $\Pi(i)$
  - With probability $1 - p - q$ add a new node and attach it to m nodes of the existing topology selected with probability $\Pi(i)$
- Repeat until the graph has the desired number of nodes n

The resulting topology might contain disconnected components.

Generalized Linear Preference (GLP)[7]

This model is an extension of the extended Barabási-Albert model which includes a parameter $\beta \in (-\infty, 1)$ enabling fine-tuning of the intensity of the preferential attachment.
Decreasing the value of β reduces the preference given to high degree nodes for attachment.
Formation process:

- Start creating a line graph of m_0 nodes
  - With probability p add m links to existing nodes randomly selected according to:

$$\Pi(i) = \frac{deg(i) - \beta}{\sum_{v \in V}(deg(v) - \beta)}$$

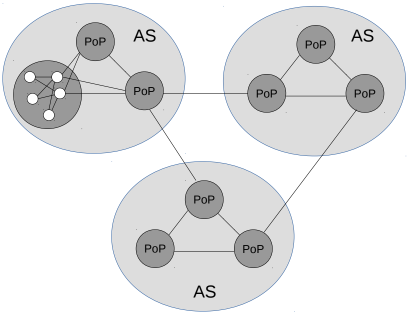
  - add a new node and attach it to m nodes of the existing topology selected with probability $\Pi(i)$
- Repeat until the graph has the desired number of nodes n

[7] T. Bu and D. Towsley. On distinguishing between internet power law topology generators. In *INFOCOM'02*

Intra-domain PoP-level or router-level topology

The network topology of a commercial ISP is a very guarded secret, mainly for security and competitive reasons

In the core network, routers are geographically clustered into Point of Presence (PoP)



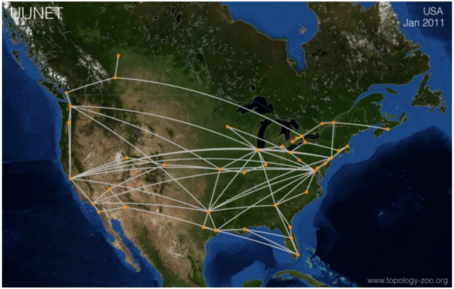
RocketFuel dataset[8]

- Dataset of PoP-level and router-level topologies inferred using Traceroute measurements from various vantage points
- Output is just network topology, no link capacities, weights or delays
- Node geolocation can be inferred by geolocating IP addresses or interpreting hostnames (e.g. `s1.umontana.nw.verio.net`)

ASN	ISP name
1221	Telstra (Australia)
1239	Sprintlink (US)
1755	EBONE (Europe)
2914	Verio (US)
3257	Tiscali (Europe)
3356	Level 3 (US)
3967	Exodus (US)
4755	VSNL (India)
6461	Abovenet (US)
7018	AT&T (US)

[8] `www.cs.washington.edu/research/projects/networking/www/rocketfuel`

Internet Topology Zoo dataset[9]



- Contains many academic networks (NRENs) but also some commercial ISPs
- Collected from various sources, mainly network maps
- Some topologies are labelled with link capacities and node locations

[9] http://www.topology-zoo.org/

Datacenter topologies

Although physical layer topologies of privately-operated datacenters are not generally publicly available, normally adopted topologies are well known.

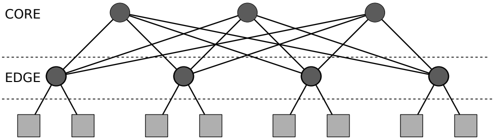
Datacenter topologies

Although physical layer topologies of privately-operated datacenters are not generally publicly available, normally adopted topologies are well known.

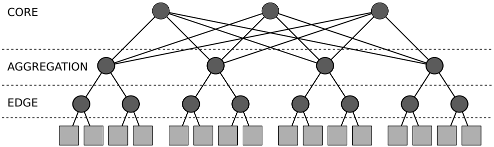
Most common topologies are:

- Two- and three-tier
- Fat tree
- B-cube

Two- and three-tier



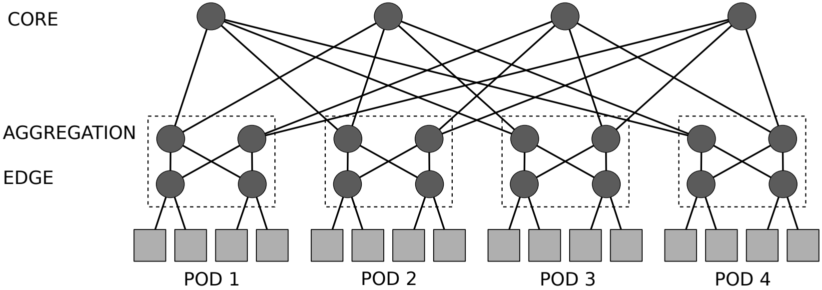
Two-tier



Three-tier

Two-tier topologies normally support up to 5K - 8K hosts, three-tier up to hundreds of thousands.

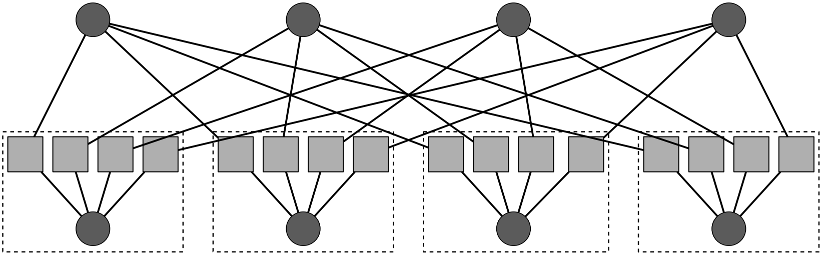
Fat tree[10]



Uses commodity switches organized in Clos-like networks.
More cost-effective than two- and three-tier.

[10] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM'08*

B-cube[11]



Specifically designed for shipping-container based, modular datacenters
Hosts are equipped with multiple network interfaces and are also used to forward traffic

[11] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: a high performance, server-centric network architecture for modular data centers. In *ACM SIGCOMM'09*

Simple models

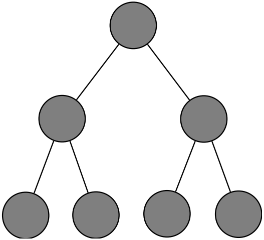
The network topologies presented so far are more-or-less accurate models of real networks.

Simple models

The network topologies presented so far are more-or-less accurate models of real networks.

However, in order to understand complex behaviors, it might be useful to adopt simpler and more tunable models providing a greater level of abstration.

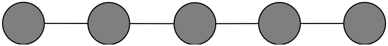
Tree



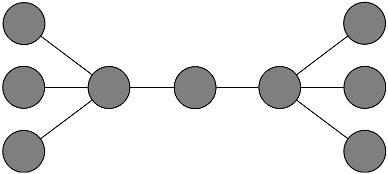
Normally used to model hierarchically organized overlays, e.g. a caching infrastructure of a Content Distribution Network (CDN).

Only one path per origin-destination pair.

Line and dumbbell



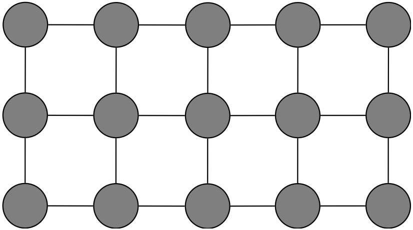
Line



Dumbbell

Normally used for basic evaluations of congestion control protocols.

Grid



It provides great shortest path diversity

$$N_{(0,0)\leftrightarrow(m,n)} = \binom{m+n}{n} = \frac{(m+n)!}{m!n!}$$

Traffic matrices

A **traffic matrix** provides, for every ingress point i into the network and every egress point j out of the network, the volume of traffic $T_{i,j}$ from i to j over a given time interval.

[12] `http://www.cs.utexas.edu/~yzhang/research/AbileneTM/`

[13] S Uhlig, B Quoitin, J Lepropre, S Balon,, Providing public intradomain traffic matrices to the research community. *ACM SIGCOMM Computer Communications Review*, 2006

Traffic matrices

A **traffic matrix** provides, for every ingress point i into the network and every egress point j out of the network, the volume of traffic $T_{i,j}$ from i to j over a given time interval.

Like topologies, ISPs treat their traffic matrices very confidentially. The only publicly available real traffic matrices are from academic networks, e.g. Abilene[12] and GEANT[13].

[12] `http://www.cs.utexas.edu/~yzhang/research/AbileneTM/`

[13] S Uhlig, B Quoitin, J Lepropre, S Balon,, Providing public intradomain traffic matrices to the research community. *ACM SIGCOMM Computer Communications Review*, 2006

Traffic matrices

A **traffic matrix** provides, for every ingress point i into the network and every egress point j out of the network, the volume of traffic $T_{i,j}$ from i to j over a given time interval.

Like topologies, ISPs treat their traffic matrices very confidentially. The only publicly available real traffic matrices are from academic networks, e.g. Abilene[12] and GEANT[13].

Fortunately, there is a way to model intradomain traffic matrices and generate them synthetically.

[12] `http://www.cs.utexas.edu/~yzhang/research/AbileneTM/`

[13] S Uhlig, B Quoitin, J Lepropre, S Balon,, Providing public intradomain traffic matrices to the research community. *ACM SIGCOMM Computer Communications Review*, 2006

Sythetic generation of traffic matrices[14]

Overview

Caveat: it can be applied only to PoP-level intradomain topologies.
It can generate the following traffic matrices:

- **Cyclostationary:** series of traffic volumes suitable for modelling diurnal patterns in network traffic

$$T_{ij}(t) = X_{ij}(t) + W_{ij}(t)$$

- **Stationary:** series of traffic volumes suitable for modelling network traffic variations over short timescales (up to 1-1.5 hours)

$$T_{ij}(t) = X_{ij} + W_{ij}(t)$$

- **Static:** traffic volumes at single point in time

$$T_{ij}(t = t_0) = X_{ij}$$

[14] A. Nucci, A. Sridharan, and N. Taft. The problem of synthetically generating IP traffic matrices: initial recommendations. *ACM SIGCOMM Computer Communication Review*, 35(3):1932, 2005.

Synthetic generation of traffic matrices

Algorithm

[15] L. Saino, C. Cocora, G. Pavlou, A Toolchain for Simplifying Network Simulation Setup, in *SIMUTOOLS'13*

Synthetic generation of traffic matrices

Algorithm

- Generate mean traffic volumes for all origin-destination pairs as:

$$X_{ij} = ln\mathcal{N}(\mu, \sigma^2)$$

[15] L. Saino, C. Cocora, G. Pavlou, A Toolchain for Simplifying Network Simulation Setup, in *SIMUTOOLS'13*

Synthetic generation of traffic matrices

Algorithm

- Generate mean traffic volumes for all origin-destination pairs as:

$$X_{ij} = ln\mathcal{N}(\mu, \sigma^2)$$

- Rank OD pairs according to *Ranking Metrics Heuristic* and map OD to traffic volumes. Further information in [15].

[15] L. Saino, C. Cocora, G. Pavlou, A Toolchain for Simplifying Network Simulation Setup, in *SIMUTOOLS'13*

Synthetic generation of traffic matrices

Algorithm

- Generate mean traffic volumes for all origin-destination pairs as:

$$X_{ij} = ln\mathcal{N}(\mu, \sigma^2)$$

- Rank OD pairs according to *Ranking Metrics Heuristic* and map OD to traffic volumes. Further information in [15].
- If dynamic traffic matrix is needed, generate random fluctuations $W_{i,j}$ as zero-mean normal random variables with standard deviation

$$\sigma_{ij} = \left(\frac{\overline{x}_{i,j}(t)}{\psi} \right)^{1/\gamma}$$

[15] L. Saino, C. Cocora, G. Pavlou, A Toolchain for Simplifying Network Simulation Setup, in *SIMUTOOLS'13*

Synthetic generation of traffic matrices

Algorithm

- Generate mean traffic volumes for all origin-destination pairs as:

$$X_{ij} = ln\mathcal{N}(\mu, \sigma^2)$$

- Rank OD pairs according to *Ranking Metrics Heuristic* and map OD to traffic volumes. Further information in [15].
- If dynamic traffic matrix is needed, generate random fluctuations $W_{i,j}$ as zero-mean normal random variables with standard deviation

$$\sigma_{ij} = \left(\frac{\overline{x}_{i,j}(t)}{\psi}\right)^{1/\gamma}$$

- If cyclostationary matrix is needed, multiply traffic volumes by a periodic function (e.g. *sin*)

[15] L. Saino, C. Cocora, G. Pavlou, A Toolchain for Simplifying Network Simulation Setup, in *SIMUTOOLS'13*

Implementing scenarios in network experiments

Now you know how to select realistic scenarios suitable for your experiments... ☺

Implementing scenarios in network experiments

Now you know how to select realistic scenarios suitable for your experiments... ☺

... But you still need to implement them ☹

Implementing scenarios in network experiments

Setting up a network simulation is a cumbersome and error-prone task. This requires to:

- select a suitable topology,
- configure it with link capacities, weights, delays, buffer sizes, protocol stacks and applications,
- generate a traffic matrix or configure traffic sources,
- deploy all this in the target simulator.

Support for scenario generation is sometimes overlooked by common network simulators and emulators.

Introducing FNSS

Fast Network Simulation Setup (FNSS) is a toolchain allowing network researchers and engineers to easily set up a network simulation scenario (topology, traffic matrix, event schedule) and deploy it in the preferred target simulator.

FNSS is made of a core library and a number of APIs and adapters:

- The core library, written in Python, generates simulation scenarios and export them to XML files.
- Adapters and APIs can be used to import a scenario in the target simulator:
  - APIs: Java, C++, Python
  - Adapters: ns-2, ns-3, Mininet, Autonetkit, Omnet++ (next release)

Topology creation

Import:

- **from other generators:** BRITE, INET, aSHIIP
- **from datasets:** RocketFuel, CAIDA AS relationships, Topology Zoo, Abilene

Generate:

- **random topologies:** Barabási-Albert, extended Barabási-Albert, Erdős-Rényi, Waxman, Generalized Linear Preference (GLP).
- **datacenter topologies:** two-tier, three-tier, fat tree, B-cube
- **simple topologies:** star, ring, line, dumbbell, tree, mesh

Topology configuration

- **Link capacities assignment:** constant, manual, random (uniform, power-law, Zipf-Mandlebrot, user-defined pdf), centrality-based models
- **Link delays assignment:** constant, manual, proportional to link length
- **Link weight assignment:** constant, manual, proportional to link delay, proportional to inverse of capacity
- **Buffer sizes assignment:** constant, manual, bandwidth-delay product, proportional to link capacity
- **Protocol stack and applications**: each node can be assigned one stacks and several applications

Topology configuration

- **Link capacities assignment:** constant, manual, random (uniform, power-law, Zipf-Mandlebrot, user-defined pdf), centrality-based models
- **Link delays assignment:** constant, manual, proportional to link length
- **Link weight assignment:** constant, manual, proportional to link delay, proportional to inverse of capacity
- **Buffer sizes assignment:** constant, manual, bandwidth-delay product, proportional to link capacity
- **Protocol stack and applications**: each node can be assigned one stacks and several applications

Topology configuration

- **Link capacities assignment:** constant, manual, random (uniform, power-law, Zipf-Mandlebrot, user-defined pdf), centrality-based models
- **Link delays assignment:** constant, manual, proportional to link length
- **Link weight assignment:** constant, manual, proportional to link delay, proportional to inverse of capacity
- **Buffer sizes assignment:** constant, manual, bandwidth-delay product, proportional to link capacity
- **Protocol stack and applications**: each node can be assigned one stacks and several applications

Topology configuration

- **Link capacities assignment:** constant, manual, random (uniform, power-law, Zipf-Mandlebrot, user-defined pdf), centrality-based models
- **Link delays assignment:** constant, manual, proportional to link length
- **Link weight assignment:** constant, manual, proportional to link delay, proportional to inverse of capacity
- **Buffer sizes assignment:** constant, manual, bandwidth-delay product, proportional to link capacity
- **Protocol stack and applications**: each node can be assigned one stacks and several applications

Topology configuration

- **Link capacities assignment:** constant, manual, random (uniform, power-law, Zipf-Mandlebrot, user-defined pdf), centrality-based models
- **Link delays assignment:** constant, manual, proportional to link length
- **Link weight assignment:** constant, manual, proportional to link delay, proportional to inverse of capacity
- **Buffer sizes assignment:** constant, manual, bandwidth-delay product, proportional to link capacity
- **Protocol stack and applications**: each node can be assigned one stacks and several applications

Topology configuration

- **Link capacities assignment:** constant, manual, random (uniform, power-law, Zipf-Mandlebrot, user-defined pdf), centrality-based models
- **Link delays assignment:** constant, manual, proportional to link length
- **Link weight assignment:** constant, manual, proportional to link delay, proportional to inverse of capacity
- **Buffer sizes assignment:** constant, manual, bandwidth-delay product, proportional to link capacity
- **Protocol stack and applications**: each node can be assigned one stacks and several applications

Traffic

Traffic

Traffic matrices: FNSS can generate synthetic traffic matrices according to the *Ranking Metrics Heuristic* method.

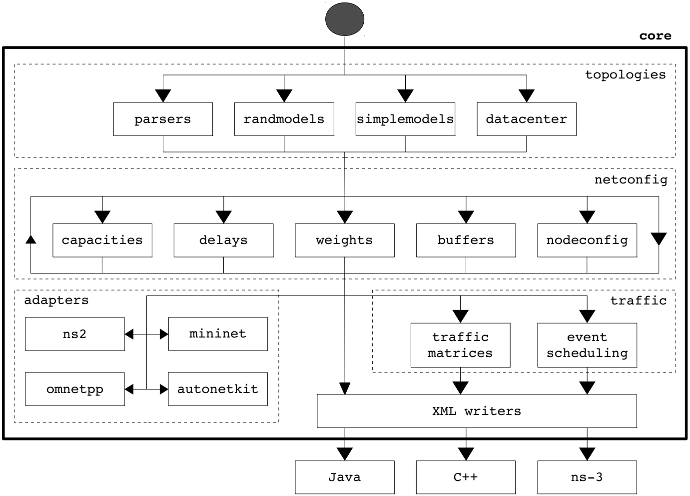
Traffic

Traffic matrices: FNSS can generate synthetic traffic matrices according to the *Ranking Metrics Heuristic* method.

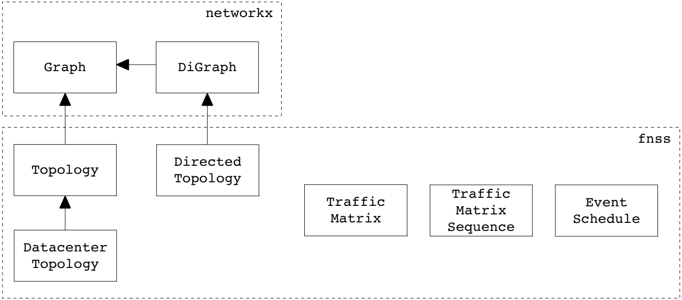
Event schedules:

- An event schedule is a list of events labelled with an execution time.
- An event is modelled as a dictionary of key-value attributes.
- The target simulator must be able to interpret the meaning of the event attributes.

Workflow


core
topologies
parsers
randmodels
simplemodels
datacenter
netconfig
capacities
delays
weights
buffers
nodeconfig
adapters
ns2
mininet
omnetpp
autonetkit
traffic
traffic matrices
event scheduling
XML writers
Java
C++
ns-3

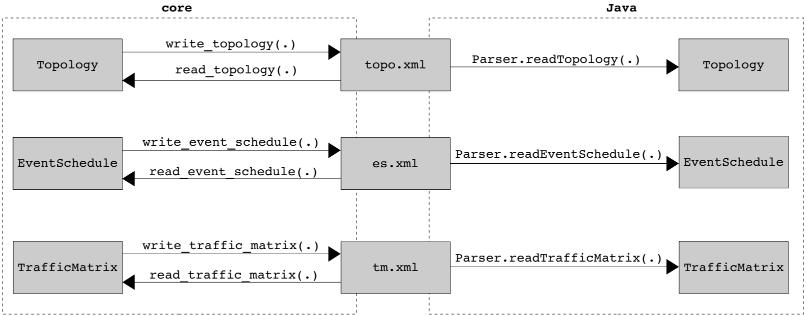
Core library



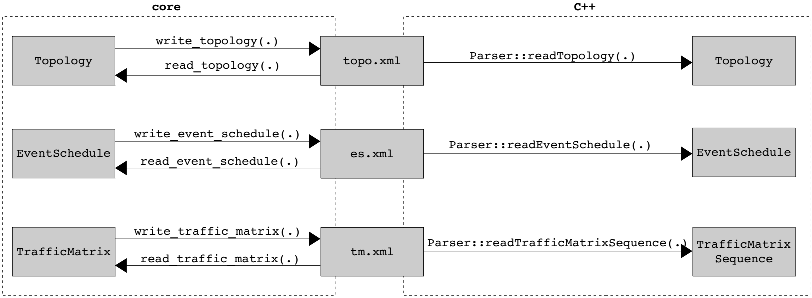
NetworkX[16] is a widely used Python library for graph analysis and visualization. Since all FNSS topology classes inherit from NetworkX `Graph`s, all NetworkX functions can be used with FNSS topologies

[16] `http://networkx.github.io`

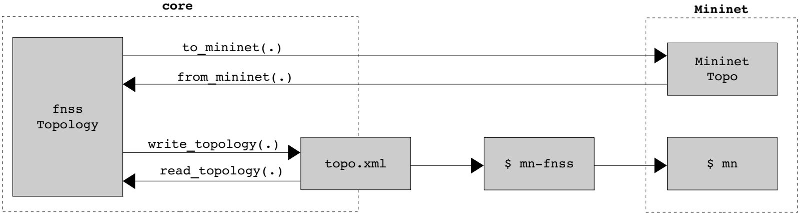
Java API


core
Java
Topology
write_topology(.)
read_topology(.)
topo.xml
Parser.readTopology(.)
Topology
EventSchedule
write_event_schedule(.)
read_event_schedule(.)
es.xml
Parser.readEventSchedule(.)
EventSchedule
TrafficMatrix
write_traffic_matrix(.)
read_traffic_matrix(.)
tm.xml
Parser.readTrafficMatrix(.)
TrafficMatrix

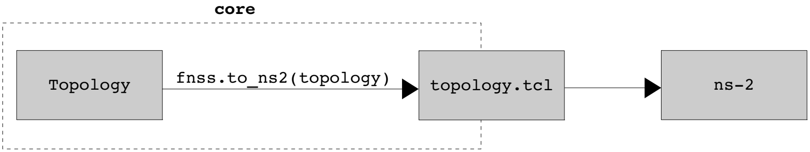
C++ API


core
C++
Topology
write_topology(.)
read_topology(.)
topo.xml
Parser::readTopology(.)
Topology
EventSchedule
write_event_schedule(.)
read_event_schedule(.)
es.xml
Parser::readEventSchedule(.)
EventSchedule
TrafficMatrix
write_traffic_matrix(.)
read_traffic_matrix(.)
tm.xml
Parser::readTrafficMatrixSequence(.)
TrafficMatrix Sequence

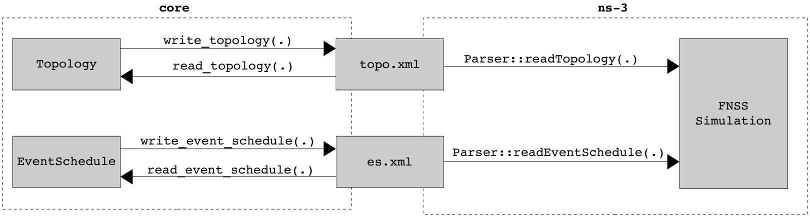
Mininet


core
Mininet
to_mininet(.)
from_mininet(.)
Mininet
Topo
fnss
Topology
write_topology(.)
read_topology(.)
topo.xml
$ mn-fnss
$ mn

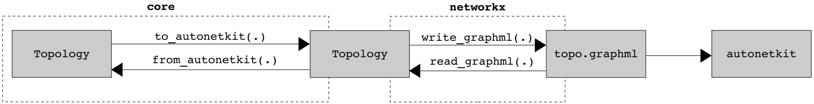
ns-2



ns-3

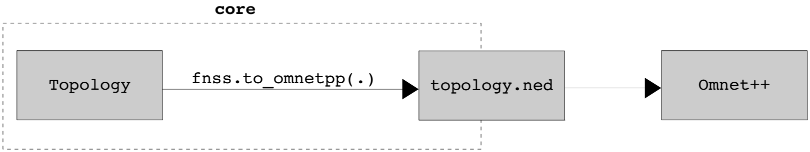

core
ns-3
Topology
write_topology(.)
read_topology(.)
topo.xml
Parser::readTopology(.)
FNSS
Simulation
EventSchedule
write_event_schedule(.)
read_event_schedule(.)
es.xml
Parser::readEventSchedule(.)

AutoNetkit[17]



[17] http://autonetkit.org/

Omnet++



Note: Currently only available on development branch

Download and installation

Let's have a look at the website!

`http://fnss.github.io`

Live coding examples

Open the terminal and type:

ipython qtconsole --pylab=inline &

Exercise 1

Do something you need!

Think of a network experiment scenario that you may use in your research. Implement it with FNSS and deploy it on the simulator/emulator of your choice.

Exercise 2

Deploy a datacenter topology to Mininet

1. Create a datacenter topology with no self-loops (*e.g.*, two-tier with 1 core switch).
2. Assign link capacities. Core-edge links: 40 Mbps, edge-server links: 10 Mbps.
3. Assign constant link delays of 2ms and buffer sizes equal to bandwidth-delay product.
4. Launch Mininet with the topology created using either `mn-fnss` or mininet's Python API.
5. Ping all nodes. Esnure all nodes are reachable.
6. Ping selected pairs of nodes. Ensure that the delays are consistent with the propagation delays you set.
7. Run iperf between a pair of nodes. Ensure that the throughput achieved is consistent with the link capacities you set.

Exercise 3

1. Download a topology of your choice from the Topology Zoo dataset.
2. Import the topology on FNSS and plot it on screen.
3. Compute number of nodes and links. What's its diameter?
4. Does it contain more than one connected component? If so, just keep the largest connected component.
5. Apply link capacities, delays, weights and buffer sizes using an algorithm of your choice.
6. Generate a dynamic (either stationary or cyclostationary) traffic matrix for the topology.
7. Export the topology to ns-2.
8. Export both topology and TM to XML files.
9. Write a Java or C++ program that parses the topology and the traffic matrix and prints on screen nodes, edges and traffic volumes.

Thank you!