Fracas
the *Fra*med *C*hannel *A*ccess *S*imulator

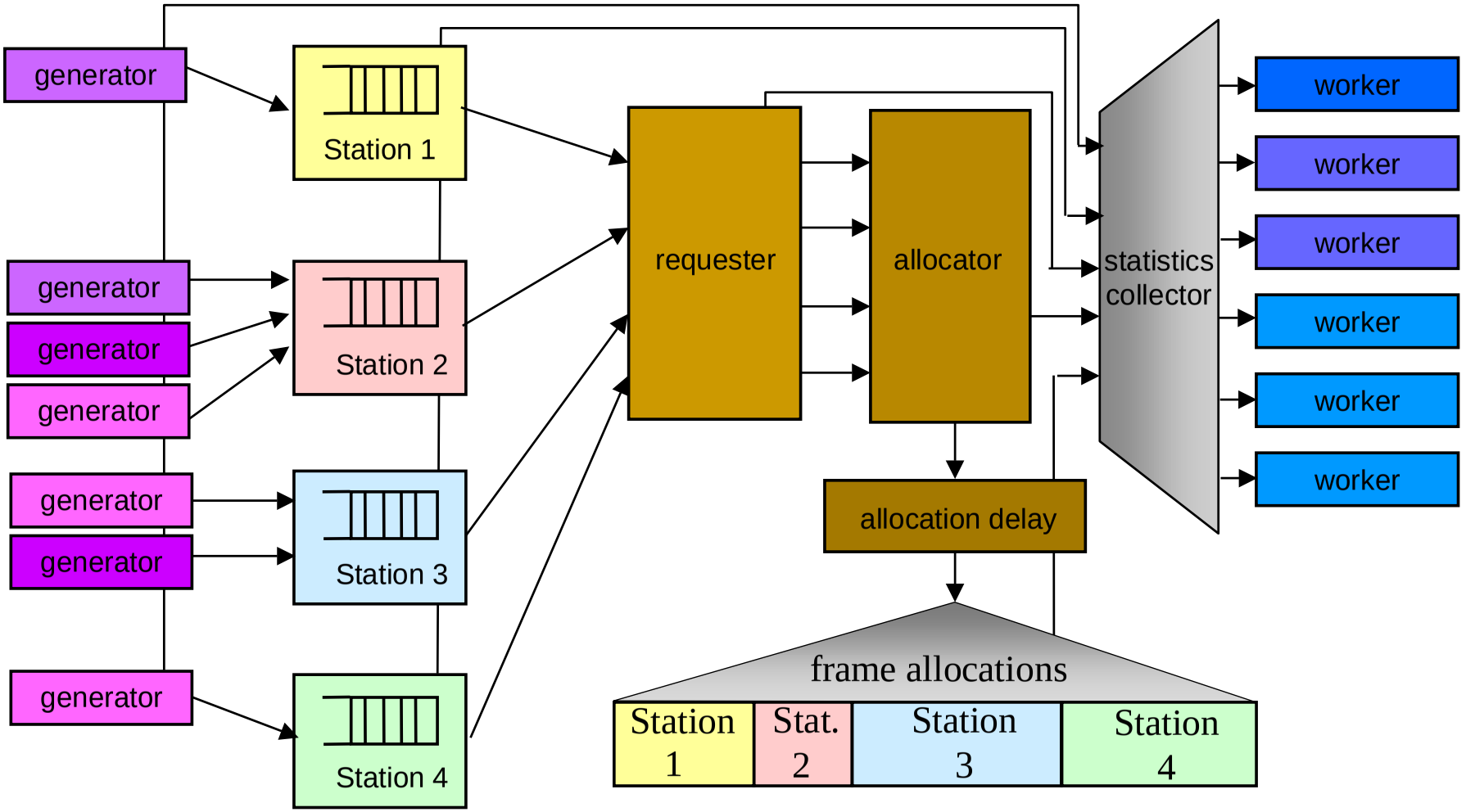
Francesco Potortì

- A very specialised simulator for communications protocols written in standard C
- Very small and very fast
- Able to study the behaviour and the performance of multiple-access protocols, usually at the MAC level
- Used until now for satellite channel access protocols
- Provides unsophisticated but comprehensive statistics
- Wrapper in Python for independent replications

Scope of the simulation

- Multiple-access methods that use a time-framed communications channel
- In principle, usable for any such type of channel: the framed channel of FDDI2, but not FDDI, unless the time is artificially subdivided in frames, putting a lower boundary on time resolution
- In practice, always used for geostationary satellite access
- The core is very small, consequently
  - the access method can be as general as possible, with only the time frame constraint
  - programming is not easy, and requires knowledge of all the inner structures of the simulator

General architecture



Traffic generators

- Each produces a number of Transmission Units per frame
- Each generator is attached to a single station
- Any number of generators can be attached to a station
- Built-in generators include:
  - two-state periodic fixed rate — can be used for one-shot
  - two-state periodic Poisson — can be used for one-shot
  - two-state Markov-modulated Poisson
  - two-dimensional (NxM states) Markov-modulated models VBR traffic
  - fractional Gaussian white noise models generic aggregate traffic
  - specialised generators model batch interactive traffic

The stations and the allocator

- All traffic is expressed as the number of Transmission Units
- TRUs are produced by generators, queued at the stations, and sent according to the allocations in the current frame
- TRUs are not received; Fracas only simulates sending
- For each frame, each station
  - Queues traffic produced by attached generators
  - Drops traffic exceeding the queue length
  - Sends queued TRUs filling the allocation in the current frame
  - Asks the allocator for allocation in a future frame
- For each frame, the allocator sets up the allocation for the future frame as TRUs available to each station

Emulator core loop

```
do {
   frame_number += 1;
   for_all_stations_do {
      input = run_generators (this_station);
      queue += input;
      sent = min (queue, allocation);
      queue -= sent;
      request = compute_request (this_station);
   }
   compute_allocations (frame_number + allocation_delay);
   gather_statistics (frame_number);
} while (! Stop_condition ());
run_workers_and_print_results ();
```

Allocator policies

- An allocator defines an allocation policy, which is an algorithm for computing the allocations for each station at a future frame based on their requests in the current frame
- Built-in policies are those that we really used in simulation work; others may be added as necessary
- Currently implemented policies:
  - fixed TDMA: a fixed assignment to each station
  - FODA/IBEA: developed at CNUCE, experimented with a prototype
  - VnL-DA: VBR allocation developed at CNUCE
  - FEEDERS: distributed allocation scheme developed at CNUCE
  - DRIFS : distributed allocation scheme developed at CNUCE
  - CFRA: developed at ENST - Toulouse (FR)

Output statistics

- At each frame, several observables are collected
- Each worker computes a different statistics on one or more observables, including:
  - TRUs input, queued, dropped, allocated, requested, sent by stations
  - allocation unused by stations
  - transmission delay, either per frame or per TRU
- Some workers produce their results at emulation time
- Others produce a result at the end of the emulation
- A Python wrapper around Fracas is used to obtain estimates of a statistics inside a given confidence interval using independent replications of the same emulator run

Classes of traffic

- Fracas distinguishes among four different classes of traffic, hierarchically ordered as follows: stream, vbr, interactive, bulk, whose names reflect their supposed usage
- When emptying the station queues, the simulation core starts from stream, and gives any unused allocation to lower-grade traffic classes
- Usage of the classes is optional; possibilities include:
  - gathering statistics for traffic generated in different classes
  - defining different allocation policies for each class
- Use of classes enables the definition of complex allocation strategies

Fracas summary

Francesco Potortì

- A lightweight, portable simulator specialised for the study of multiple-access allocation schemes
- Traffic generators, allocation policies and statistics computations can each be added as separate modules
- Many built-in modules are implemented and have been used in research work
- Fracas has been validated by checking against a prototypal implementation of the FODA/IBEA allocation algorithm
- A paper on the architecture of Fracas has been published on Telecommunications Systems in 1999