Ingineria Programării în Reţea (IPR)

Dimensionarea cozilor

Cozile

- Structuri de date de tip FIFO (First In First Out)
- În cazul aplicaţiilor de reţea: memorii (buffere) pentru stocarea (temporară, intermediară) sau recepţionarea pachetelor
- Există şi cozi de trimitere!

De ce avem nevoie de cozi?

- *Ele se vor forma oricum*, atunci când rata de sosire a pachetelor unei aplicaţii, la un dispozitiv de reţea sau chiar destinaţie, este mai mare decât rata de plecare sau de primire a acestora
- Avem nevoie de ele pentru a reduce pierderile de pachete (în anumite limite)
- *Întrebare*: cât de mari să fie aceste cozi? Câte resurse (memorie) alocăm pentru ele?

Modele probabilistice

- Modelele MM1 şi MM1K
- Conform notaţiei Kendall:
  - Primul M = timpul între sosiri consecutive (pachete, clienţi) are o distribuţie exponenţială
  - Al doilea M = timpul de servire are o distribuţie exponenţială
  - 1 = un singur server care deserveşte coada de aşteptare
  - K = capacitatea sistemului (lungimea cozii)

Modelul MM1

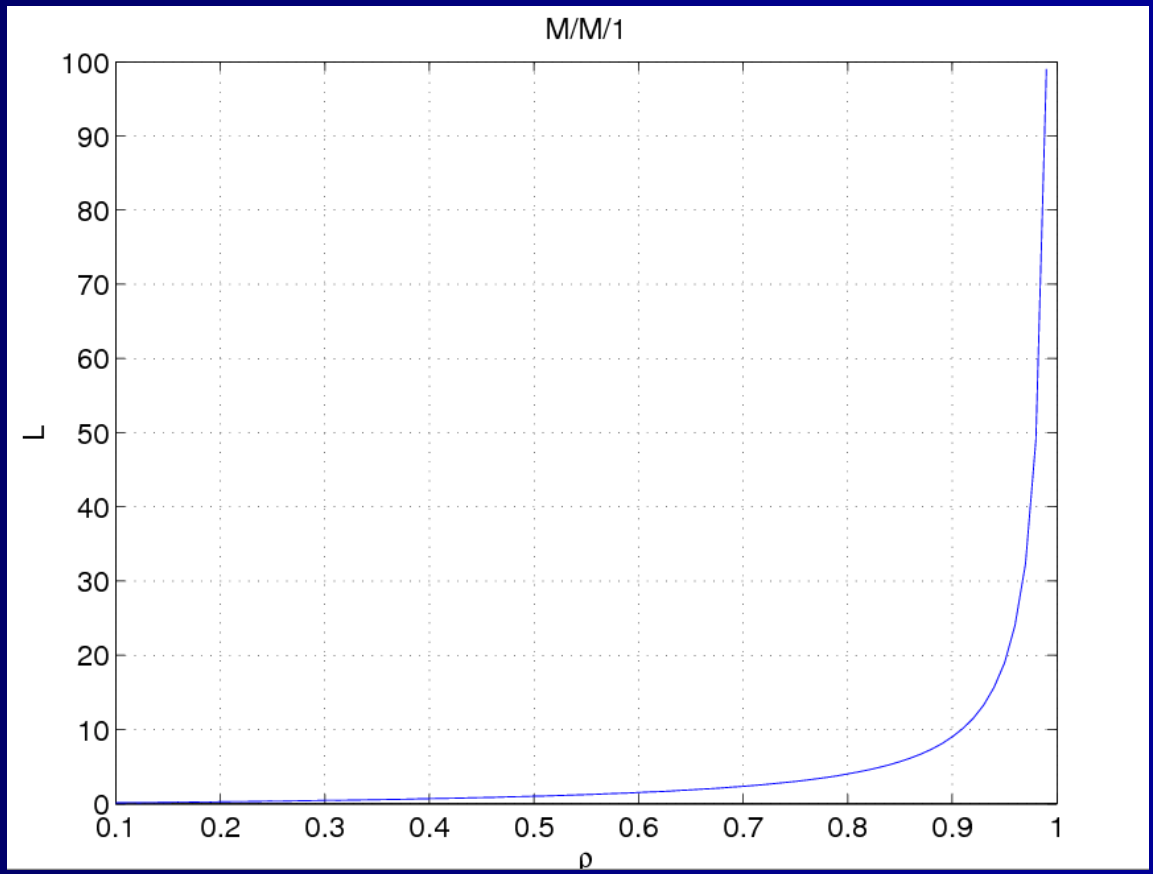
- Probabilitatea de a avea *n* pachete în coadă (ρ este încărcarea sistemului, raportul dintre rata de intrare şi cea de ieşire):

$$p_n = (1-\rho)\rho^n$$

- Timpul de aşteptare (proporţional cu numărul de pachete din coadă)

$$L = \frac{\rho}{1-\rho}$$

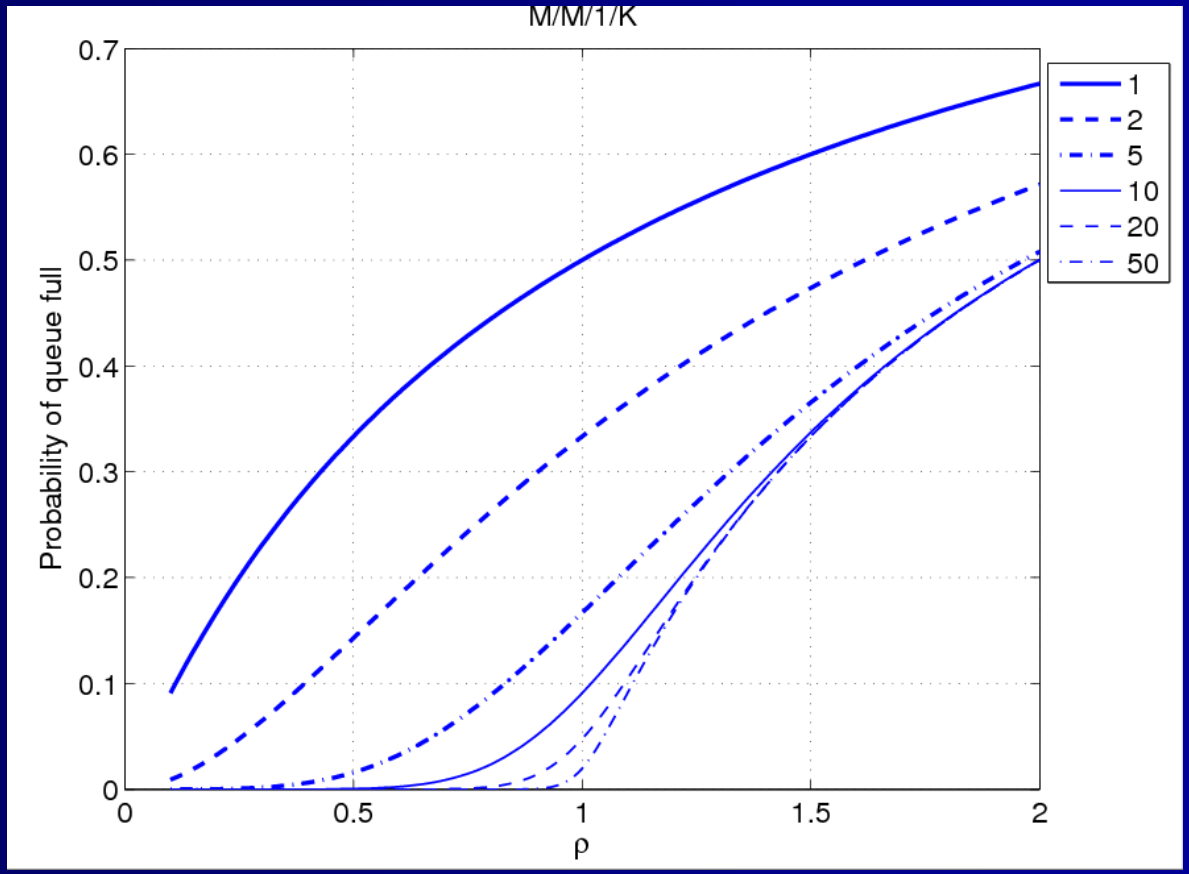
MM1 - întârzierea



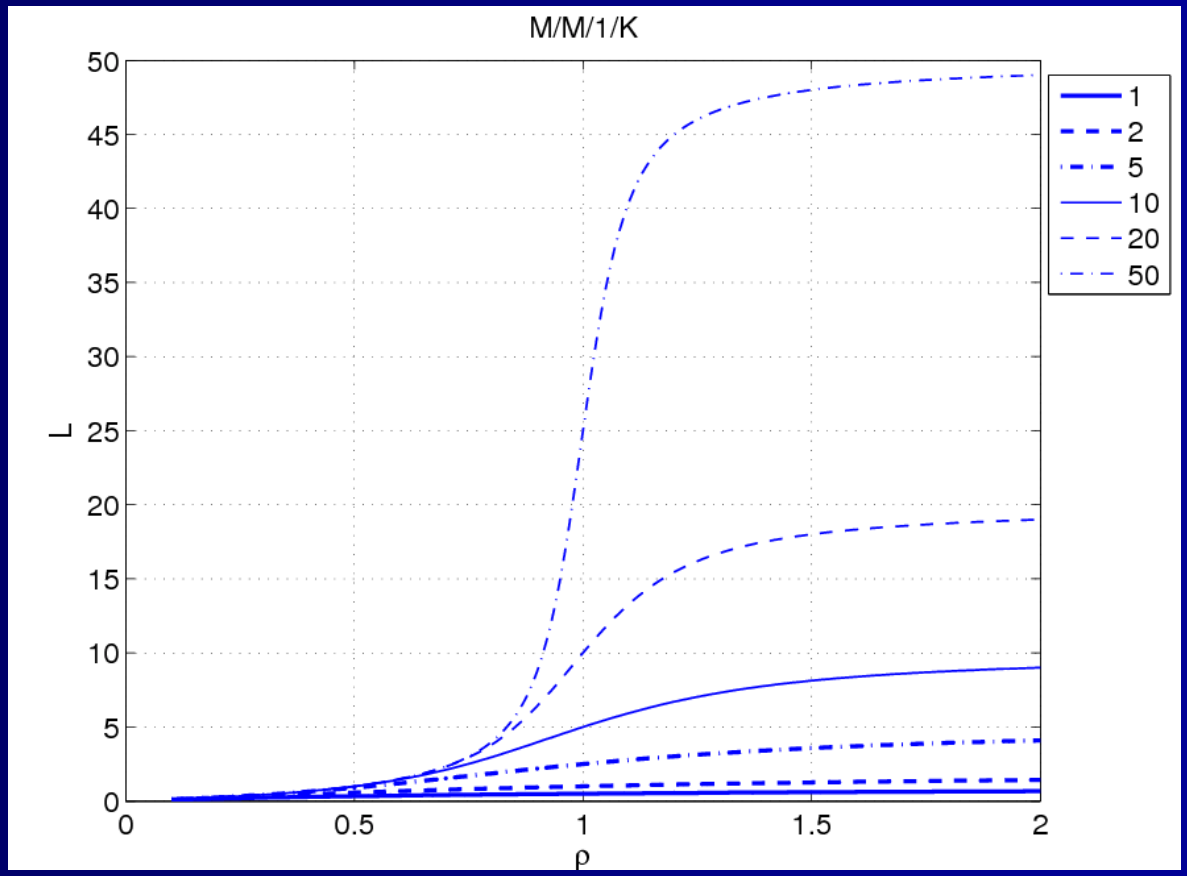
Modelul MM1K

- MM1 – reprezintă modelul pentru o coadă infinită (nu există pierderi, doar întârzieri)
- MM1K – spaţiul din coadă este limitat, prin urmare vor apărea pierderi de pachete atunci când ocuparea cozii este de 100%

MM1K - pierderi



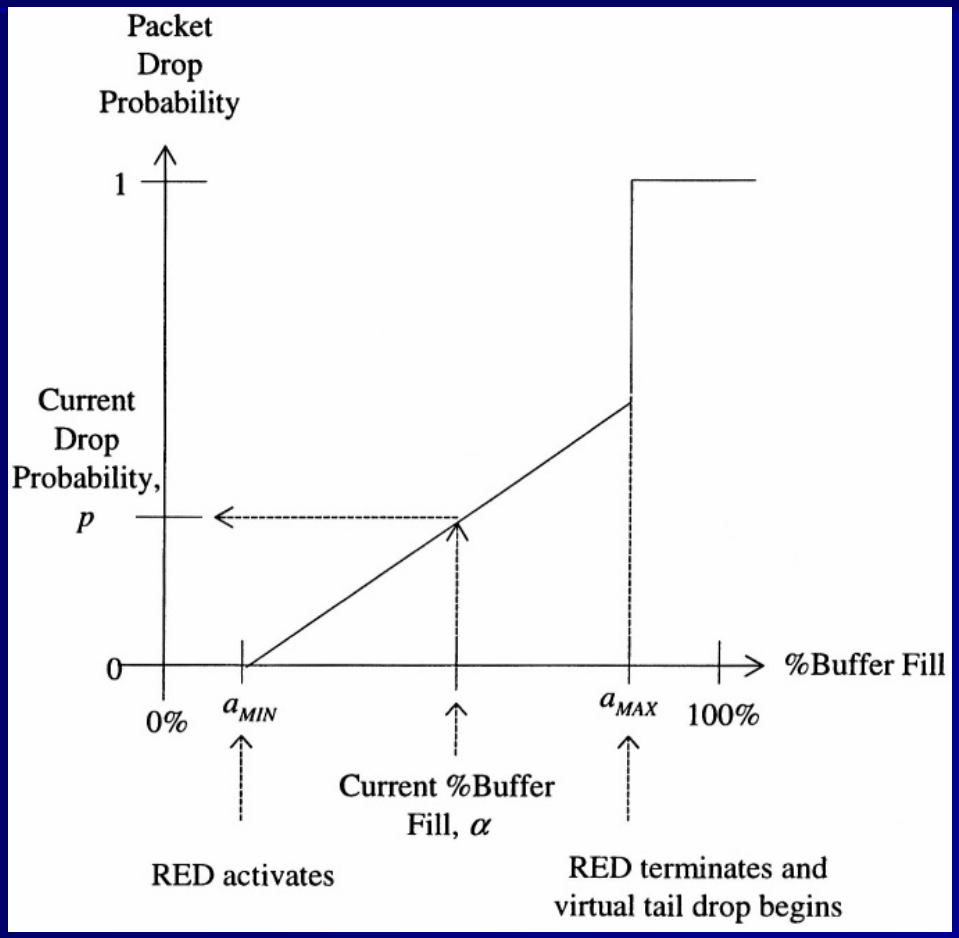
MM1K - întârzieri



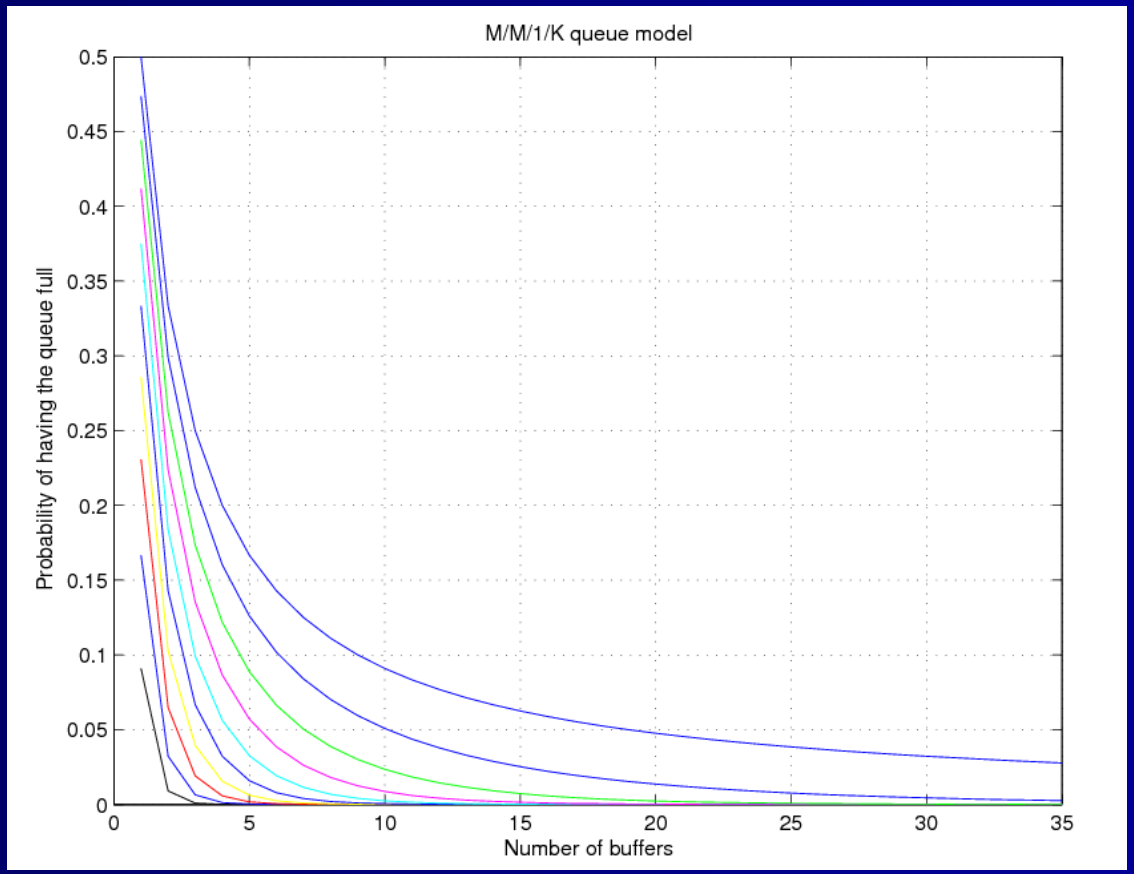
MM1K

- Coadă “prea scurtă”:
  - Pierderi mari (dezavantaj)
  - Întârzieri mici (avantaj)
- Coadă “prea lungă”:
  - Pierderi mici (avantaj)
  - Întârzieri mari (dezavantaj)

Random Early Discard (RED) (Stuff/books2)



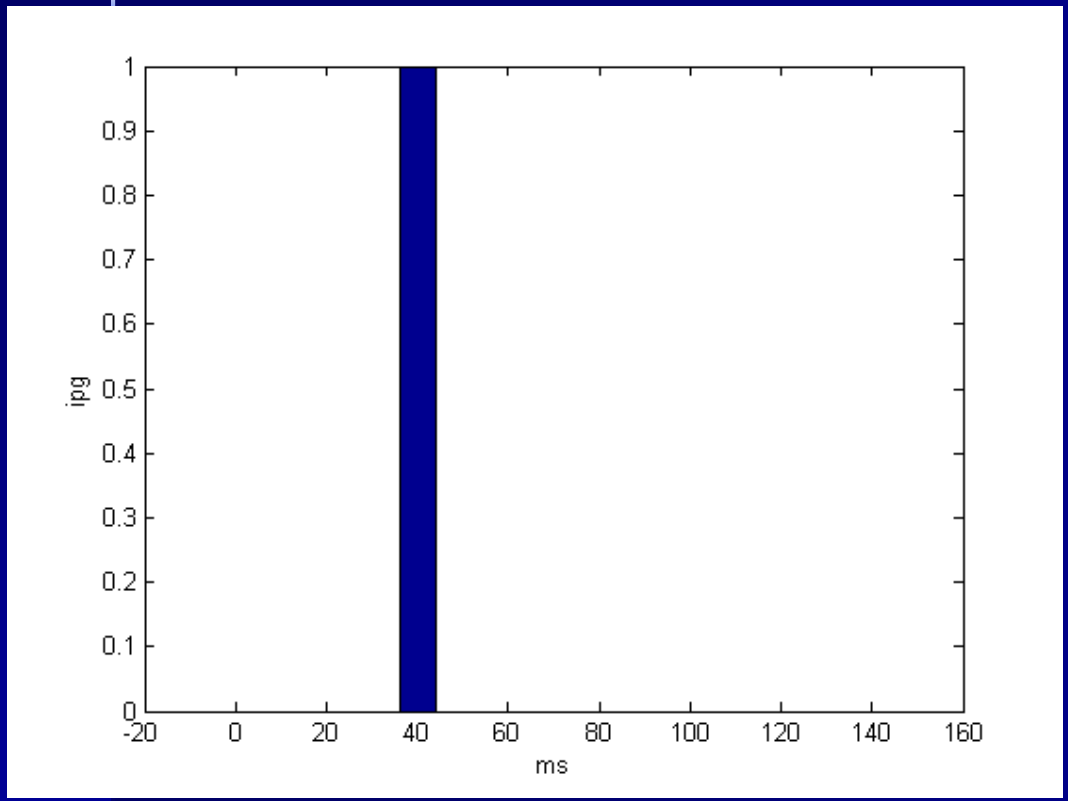
UDP – caz generic MM1K



TCP

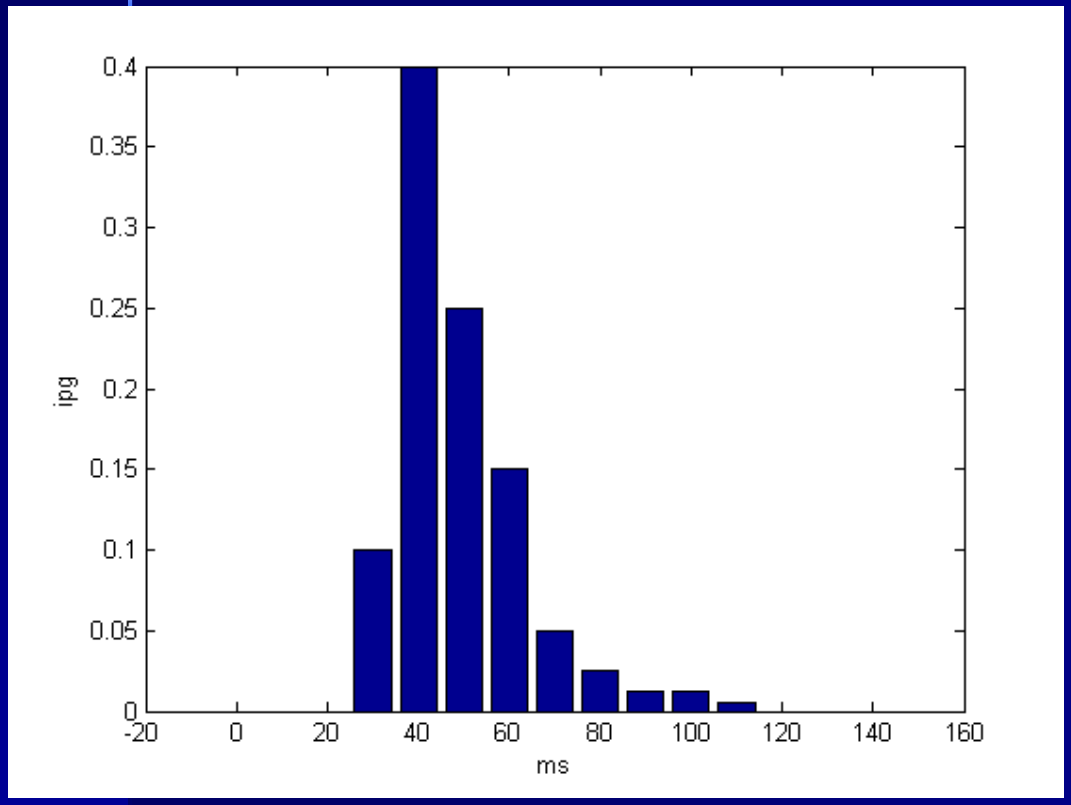
- Dimensiunea optimă a cozilor de transmisiune/recepţie:
  W = BDP (Bandwidth Delay Product)
- Asigură rata maximă de transmisiune (implicit ocuparea completă a benzii disponibile)

VoIP – G.711



- La "sender"
  - IPG = 40ms (inter-packet gap)
  - constant

VoIP (2)



- La "receiver"
  - Jitter-ul va afecta IPG-ul, conform cu histograma din imagine
  - Valoarea maximă a IPG-ului = 120ms

VoIP – dejittering buffer

- La recepţie, pentru anularea efectelor jitter-ului, este nevoie de o coadă (buffer de dejittering) de lungime:

$$coada = \frac{ipg_{\max}}{ipg_{sender}} = \frac{120ms}{40ms} = 3\,pachete$$

Simularea cozilor (Python)

```
from __future__ import generators
from SimPy.Simulation import *
import random
import math
import sys

# Random number generator with a Poisson distribution
def RandNegExp( mean ): ### {{{
    x = random.random()
    while x == 0:
        x = random.random()
    return ( - long( round( mean * math.log( x ) ) ) )
### }}}
```

Clasa “packet”

```
class Packet: ### {{{
def __init__( self, generator_id, packet_size,
    departure_time, sequence_number ):
    self.id = generator_id # identifier of the source that
          generated the packet
    self.size = packet_size # the size of the packet
    self.departure = departure_time
    self.sequence_number = sequence_number
    self.experienced_delay = 0
### }}}
```

Clasa "generator de pachete"

```
class PacketGenerator( Process ): ### {{{
    def __init__( self, id, av_ipg, av_size, no_packets ):
        Process.__init__( self )
        self.id = id
        self.av_ipg = av_ipg
        self.av_size = av_size
        self.no_packets = no_packets
        self.sequence_number = 1
        self.index = 0
        self.trace_file = open( "departures.txt", "w" )
        self.size_file = open( "sizes.txt", "w" )
```

```
def generate_packet( self, Queue ):
        while self.index < self.no_packets:
        size = RandNegExp( self.av_size ) # generate the size
        while size == 0:
                size = RandNegExp( self.av_size )
                pkt = Packet( self.id, size, now(), self.sequence_number )
                message = "%7.2f\n" % pkt.size
                self.size_file.write( message )

        # enqueue the packet
        yield request, self, sharedResource
        Queue.enqueue( pkt )
        yield release, self, sharedResource
        message = "%7.2f\n" % now()
        self.trace_file.write( message )
        self.sequence_number += 1
        self.index += 1
        # wait before generating the next packet
        waittime = RandNegExp( self.av_ipg )
        yield hold, self, waittime
```

Clasa “coadă infinită”

```
class InfiniteQueue: ### {{{
    def __init__( self ):
        self.queue = []
    def enqueue( self, packet ):
        self.queue.append( packet )
    def dequeue( self ):
        if len( self.queue ) != 0:
            rqueue = self.queue
            rqueue.reverse()
            packet = rqueue.pop()
            rqueue.reverse()
            self.queue = rqueue
            return packet
        else:
            return 0
```

Clasa “server”

```
class Server( Process ): ### {{{
    def __init__( self, no_packets ):
            Process.__init__( self, no_packets )
            self.no_packets = no_packets
            self.index = 0
            self.arrivals_file = open( "arrivals.txt", "w" )
            self.delays_file = open( "delays.txt", "w" )
    def service_packet( self, Queue ):
            while self.index < self.no_packets:
            yield request, self, sharedResource
            packet = Queue.dequeue() # dequeue a packet
            yield release, self, sharedResource
```

```
if packet != 0:
        service_time = packet.size
        yield hold, self, service_time # service the packet
        packet.experienced_delay = now() - packet.departure
        message = "%7.2f\n" % now()
        self.arrivals_file.write( message )
        message = "%7.2f\n" % packet.experienced_delay
        self.delays_file.write( message )
        self.index += 1
else:
        yield hold, self, 0.001 ### take a short nap before
                                ### requesting access to the queue
```

Programul principal

```
id = 0
av_size = 10 # size = service time; av_size = mu
av_ipg = 5 # av_ipg = lambda
no_packets = 1000
end_time = 100000

# create an object for the infinite queue
q = InfiniteQueue()

# semaphore for the access to the queue
sharedResource = Resource( capacity = 1 )
initialize()
```

```
# instantiate the packet generator and active its generation function
p = PacketGenerator( id, av_ipg, av_size, no_packets )
activate( p, p.generate_packet( q ), at = 0.0 )

# create the server
s = Server( no_packets )
activate( s, s.service_packet( q ), at = 0.0 )

# run the simulation
simulate( until = end_time )
```