Multicast problem

In [31]:
```
import sys
import numpy as np
import gamspy as gp
import gamspy.math as gpm
from gamspy import Sum, Card
```

In [32]:
```
class arguments:
  def __init__(self, solver='cplex', DEMAND=10, N=20, M=4):
    self.solver = solver
    self.DEMAND = DEMAND
    self.N = N
    self.M = M

args = arguments()

options = gp.Options(absolute_optimality_gap=0,relative_optimality_gap=1e-6,
cont = gp.Container(options=options)

i = cont.addSet('i',description='break points',records=range(0,4))
seg = cont.addSet('seg',domain=[i],description='price break segments',record
n = cont.addSet('n',description='nodes',records=range(1,args.N+1))
# define arcs for cost definition
j = cont.addAlias('j',n)
k = cont.addAlias('k',n)
s = cont.addSet('s',domain=[j,k],description='arcs')
s[j,k].where[gpm.abs(j.ord-k.ord) == 1] = True

# define abscissae and ordinates of piecewise linear function
# segments: 0->5, 5->20, 20->100
B = cont.addParameter('BR',domain=i,description='Breakpoints (quantities at
        records=np.array([0, 5, 20, 100]))
CBR = cont.addParameter('CBR',domain=i,description='Function value at break
CBR[i] = gpm.log(B[i]+1)

numServers = cont.addParameter('numservers',description='limit on the number
# define demands at the nodes - uniform in this example
demand = cont.addParameter('demand',domain=[n])
demand[n] = (2*n.ord/Card(n))*args.DEMAND

bigM = cont.addParameter('bigM')
bigM[:] = Sum(n, demand[n])
```

In [33]:
```
# Convert function to slope intercept
m = cont.addParameter('m',domain=[j, k, i],description='gradient on segment'
m[s[j, k], i].where[seg[i]] = (CBR[i]-CBR[i.lag(1)])/(B[i]-B[i.lag(1)])

c = cont.addParameter('c',domain=[j, k, i],description='intercept cost on se
c[s[j, k], i].where[seg[i]] = CBR[i.lag(1)] - m[j, k, i]*B[i.lag(1)]
```

In [47]:
```
# enter model here
flow = cont.addVariable('flow','positive',domain=[n],description='flow from
arc_flow = cont.addVariable('arc_flow','positive',domain=[j, k],description=
b = cont.addVariable('b', 'binary' ,domain=[j, k, i],description='use piece
z = cont.addVariable('z','binary',domain=[n],description='use node n')
w = cont.addVariable('w','free',domain=[j, k, i],description= 'flow to node


#Create a flow balance constraint meeting demand
balance = cont.addEquation('balance', domain=[n])
balance[n] = (Sum(j.where[s[j, n]], arc_flow[j, n]) - (Sum(k.where[s[n, k]],

#Have a constraint for upper bound on servers
max_servers = cont.addEquation('max_servers')
max_servers[:] = Sum(n, z[n]) <= numServers

#Create a constraint for upper bound on flow
flow_bound = cont.addEquation('flow_bound', domain=n)
flow_bound[n] = flow[n] <= bigM * z[n]

#Ensure each arc is equal to each flow
arcflow = cont.addEquation('arcflow', domain=[j,k])
arcflow[s[j,k]] = Sum(seg, w[j, k, seg]) == arc_flow[j, k]

#PW Linear Equations Below

## Bounds on the flow constraints
wlo = cont.addEquation('flowC1', domain=[j, k, seg])
wlo[s[j,k], seg] = B[seg.lag(1)] * b[j,k,seg] <= w[j,k,seg]

wup = cont.addEquation('flowC2', domain=[j, k, seg])
wup[s[j,k], seg] = w[j, k, seg] <= B[seg] * b[j,k,seg]

#One piece constaint from 26linear
OnePiece = cont.addEquation('OnePiece',domain=[j, k])
OnePiece[s[j, k]]= Sum(seg, b[j, k, seg]) == 1



netPW = cont.addModel('netPW',
    equations=cont.getEquations(),
    problem=gp.Problem.MIP,
    sense=gp.Sense.MIN,
    objective=Sum((s[j, k], seg), c[j, k, seg] * b[j, k, seg] + m[j, k, seg]
)

netPW.solve(solver=args.solver, solver_options={'numericalemphasis': 1})
```

Out[47]:

	Solver Status	Model Status	Objective	Num of Equations	Num of Variables	Model Type	Solver	Sol T
0	Normal	IntegerInfeasible	NA	346	307	MIP	CPLEX	0.

In [48]:
```
print(f"Cost = {netPW.objective_value}, time = {netPW.total_solver_time}")
print(f"Use segment =\n {b.pivot()}\nlamda =\n {w.pivot()}")
```

```
print(f"Use server =\n {z.records.set_index('n')}")
print(flow.records)
```

```
Cost = 2e+300, time = 0.029000546783208847
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[48], line 2
      1 print(f"Cost = {netPW.objective_value}, time = {netPW.total_solver_t
ime}")
----> 2 print(f"Use segment =\n {b.pivot()}\nlamda =\n {w.pivot()}")
      3 print(f"Use server =\n {z.records.set_index('n')}")
      4 print(flow.records)

File ~/CS524/venv/lib/python3.10/site-packages/gams/transfer/syms/_mixins/pi
vot.py:237, in PivotVariableMixin.pivot(self, index, columns, value, fill_va
lue)
    217 """
    218 Convenience function to pivot records into a new shape (only symbols
with >1D can be pivoted)
    219
   (...)
    234     Pivoted records dataframe
    235 """
    236 # check & set
--> 237 index, columns = super().pivot(index, columns)
    239 #
    240 # ARG: value
    241 # set defaults
    242 if not isinstance(value, (str, type(None))):

File ~/CS524/venv/lib/python3.10/site-packages/gams/transfer/syms/_mixins/pi
vot.py:50, in PivotBase.pivot(self, *args)
     48 # set defaults
     49 if index is None:
---> 50     index = self.records.columns[: self.dimension - 1].tolist()
     52 if isinstance(index, str):
     53     index = [index]

AttributeError: 'NoneType' object has no attribute 'columns'
```

In [ ]: