Performance of Networked Systems

Assignment 1

Daniele Di Cesare
VU Amsterdam
d.dicesare@student.vu.nl

Gabriel Marica
VU Amsterdam
g.marica@student.vu.nl

Lecturer:
Prof.dr. Rob van der Mei

November 30, 2025

Abstract

This document is the submission of Daniele Di Cesare and Gabriel Marica for the first assignment of Performance of Networked Systems. This assignment is divided into four sections.
I. Planning of cellular telephone networks with video-conferencing services
This section involves applying traffic models to cellular networks with multiple classes.
II. Optimal distribution of channels over neighboring cells in mobile voice networks
This section focuses on channel distribution that minimizes the overall blocking probability.
III. Traffic Management in IP networks
This section shows the effects of Traffic Shaping and Traffic Policing on an incoming traffic stream.
IV. Performance of TCP-based networks
The final section addresses the performance implications of TCP Slow Start.

Contents

I. Planning of cellular telephone networks with video-conferencing services 3
1. Problem Modeling 3
2. Blocking Probability 3
3. Parameter Modifications 3
4. Multi-Rate Model Markov Chain 3
5. Balance Equations 4
6. Product-Form solution 6
7. Blocking Probability 6
8. Kaufman-Roberts Solution 6
9. Kaufman-Roberts Recursion in Python 8
II. Optimal distribution of channels over neighboring cells in mobile voice networks 10
10. Call attempt probabilities 10
11. Optimal distribution of channels 10
12. Optimal distribution below 1% 12
III. Traffic Management in IP networks 14
13. Traffic Shaping Delay Graph 14
14. Traffic Shaping Bitrate Graph 14
15. Traffic Policing Bitrate Graph 15
16. Traffic Policing Conforming Graph 15
IV. Performance of TCP-based networks 17
17. Downside of TCP Slow Start 17
18. Slow Start Transfer Times 17

I. Planning of cellular telephone networks with video-conferencing services

1. Problem Modeling

Assumptions:
1. Cell area is A
2. Number of lines is N
3. Spatial arrival intensity is λ'
4. Poisson process of "calls" with rate λ
5. Average call duration is β

Parameters:
1. $A = 1.2\text{Km}^2$
2. $N = 4$
3. $\lambda' = \frac{25}{\text{h}\cdot\text{Km}^2}$
4. $\lambda = \lambda' A = \frac{25}{\text{h}\cdot\text{Km}^2} 1.2\text{Km}^2 = 30\frac{1}{\text{h}}$
5. $\beta = \frac{1}{12}\text{h}$

We want to find the blocking probability.

2. Blocking Probability

Using Erlang Blocking formula we know that the probability of having k blocked channels is:

$$\pi_k = \frac{(\lambda\beta)^k/k!}{\sum_{i=0}^{N}(\lambda\beta)^i/i!} \tag{1}$$

Since N = 4 and $\lambda\beta = 2.5$ we get that the blocking probability of a call is:

$$\begin{aligned} \pi_N &= \frac{(\lambda\beta)^N/N!}{\sum_{i=0}^{N}(\lambda\beta)^i/i!} \\ &= \frac{2.5^4/24}{1 + 2.5 + \frac{2.5^2}{2} + \frac{2.5^3}{6} + \frac{2.5^4}{24}} \\ &\approx 0.15 \end{aligned} \tag{2}$$

So a call will be blocked 15% of the time.

3. Parameter Modifications

If the call arrival rate triples, while the average call duration becomes three times as small the blocking probability stays the same.

We can immagine each call occuping a slot of random size β, with this change the slot is divided into three equal parts $\frac{\beta}{3}$, and each part is used by a different caller since the rate tripled.

An analogy would be a leaky bucket (more on that later :P). If you triple how often you pour water but each time you pour a third the water the average water level will stay the same. So the probability that the bucket is full stays the same.

4. Multi-Rate Model Markov Chain

Assumptions:
1. Cell area is A
2. Number of lines is N

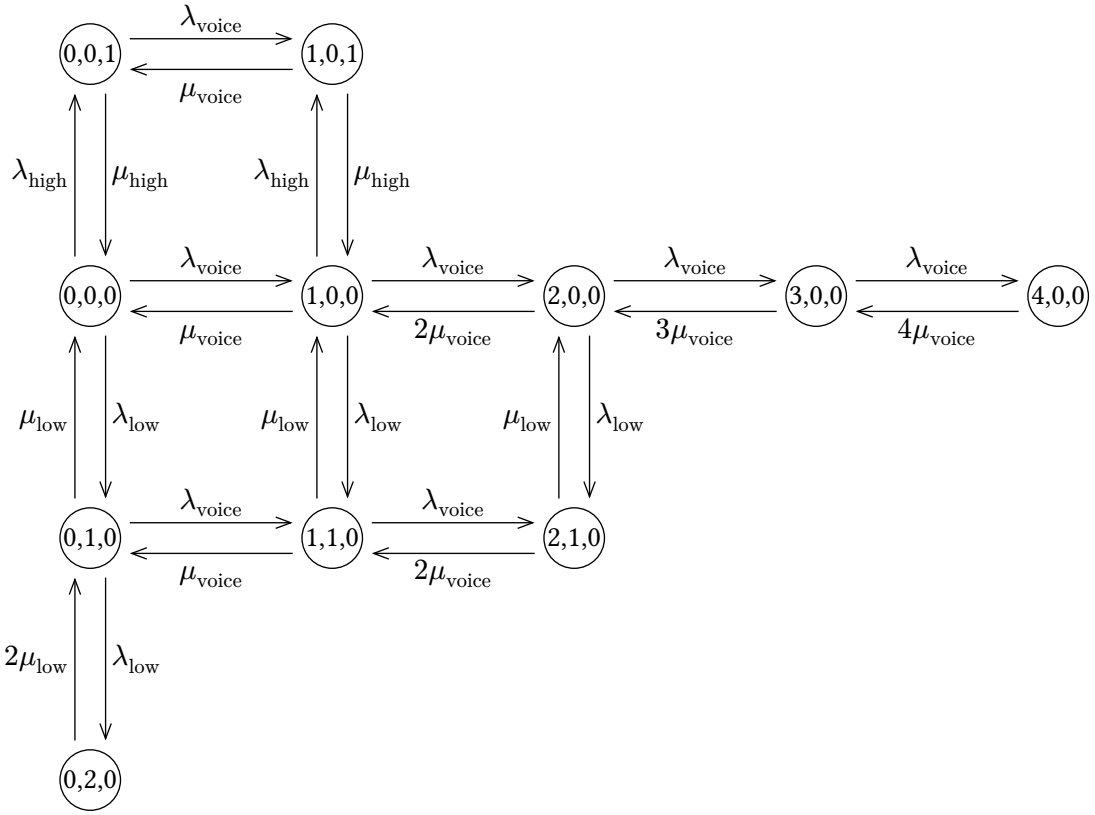
3. Probability of a low-resolution video call is P_{low}
4. Spatial arrival intensity for voice call is ${\lambda_{\text{voice}}}'$
5. Spatial arrival intensity for video call is ${\lambda_{\text{video}}}'$
6. Poisson processes of "calls" with rate $\left(\lambda_{\text{voice}}, \lambda_{\text{low}}, \lambda_{\text{high}}\right)$
7. Average call duration is $\left(\beta_{\text{voice}}, \beta_{\text{low}}, \beta_{\text{high}}\right)$

Parameters:
1. $A = 1.2\text{Km}^2$
2. $N = 4$
3. $P_{\text{low}} = 80\%$
4. $\lambda_{\text{voice}} = 30\frac{1}{\text{h}}$ (same as before)
5. $\beta_{\text{voice}} = \frac{1}{12}\text{h}$
6. ${\lambda_{\text{video}}}' = \frac{0.8}{\text{h}\cdot\text{Km}^2}$
7. $\lambda_{\text{low}} = {\lambda_{\text{video}}}' A P_{\text{low}} = \frac{0.8}{\text{h}\cdot\text{Km}^2} \cdot 1.2\text{Km}^2 \cdot 0.8 = \frac{96}{125}\frac{1}{\text{h}}$
8. $\lambda_{\text{high}} = {\lambda_{\text{video}}}' A (1 - P_{\text{low}}) = \frac{0.8}{\text{h}\cdot\text{Km}^2} \cdot 1.2\text{Km}^2 \cdot 0.2 = \frac{24}{125}\frac{1}{\text{h}}$
9. $\beta_{\text{low}} = \beta_{\text{high}} = \frac{3}{10}h$

The state space is:

$$S = \left\{n = \left(n_{\text{voice}}, n_{\text{low}}, n_{\text{high}}\right) \in I^3 : n_{\text{voice}} + 2n_{\text{low}} + 3n_{\text{high}} \leq 4\right\} \tag{3}$$

The evolution of the number of calls $n = \left(n_{\text{voice}}, n_{\text{low}}, n_{\text{high}}\right)$ can be represented by the following Markov-chain:



5. Balance Equations

Note that in this section voice, low and low were abbriviated to v, l, h, to make the formulas fit in a single line.

We know that in equilibrium the rate-in must equal the rate-out. Using the transitions of the Markov-chain we can derive the following balance equations:

$$(\lambda_v + \lambda_l + \lambda_h)\pi(0,0,0) = \mu_v\pi(1,0,0) + \mu_l\pi(0,1,0) + \mu_h\pi(0,0,1) \tag{B.1}$$

$$(\lambda_v + \lambda_l + \lambda_h + \mu_v)\pi(1,0,0) = 2\mu_v\pi(2,0,0) + \mu_l\pi(1,1,0) + \mu_h\pi(1,0,1) + \lambda_v\pi(0,0,0) \tag{B.2}$$

$$(\lambda_v + \lambda_l + 2\mu_v)\pi(2,0,0) = 3\mu_v\pi(3,0,0) + \mu_l\pi(2,1,0) + \lambda_v\pi(1,0,0) \tag{B.3}$$

$$(\lambda_v + 3\mu_v)\pi(3,0,0) = 4\mu_v\pi(4,0,0) + \lambda_v\pi(2,0,0) \tag{B.4}$$

$$4\mu_v\pi(4,0,0) = \lambda_v\pi(3,0,0) \tag{B.5}$$

$$(\lambda_v + \lambda_l + \mu_l)\pi(0,1,0) = \mu_v\pi(1,1,0) + 2\mu_l\pi(0,2,0) + \lambda_l\pi(0,0,0) \tag{B.6}$$

$$(\lambda_v + \mu_v + \mu_l)\pi(1,1,0) = 2\mu_v\pi(2,1,0) + \lambda_v\pi(0,1,0) + \lambda_l\pi(1,0,0) \tag{B.7}$$

$$(2\mu_v + \mu_l)\pi(2,1,0) = \lambda_v\pi(1,1,0) + \lambda_l\pi(2,0,0) \tag{B.8}$$

$$2\mu_l\pi(0,2,0) = \lambda_l\pi(0,1,0) \tag{B.9}$$

$$(\lambda_v + \mu_h)\pi(0,0,1) = \mu_v\pi(1,0,1) + \lambda_h\pi(0,0,0) \tag{B.10}$$

$$(\mu_v + \mu_h)\pi(1,0,1) = \lambda_v\pi(0,0,1) + \lambda_h\pi(1,0,0) \tag{B.11}$$

Remember also that the sum of the probabilities has to be 1. To ensure that we find only one solution for the balance equations we will discard Balance Equation B.2 since it is the longhest. So we also need to impose that:

$\pi(0,0,0) + \pi(1,0,0) + \pi(2,0,0) + \pi(3,0,0) + \pi(4,0,0) + \pi(0,1,0) + \pi(1,1,0) + \pi(2,1,0) + \pi(0,2,0) + \pi(0,0,1) + \pi(1,0,1) = 1$

Now we have a system of 11 equations and 11 unknowns that we need to solve using a matrix solver.

$$\begin{pmatrix}
\lambda_v+\lambda_l+\lambda_h & -\mu_v & 0 & 0 & 0 & -\mu_l & 0 & 0 & 0 & -\mu_h & 0 \\
0 & -\lambda_v & \lambda_v+\lambda_l+2\mu_v & -3\mu_v & 0 & 0 & 0 & -\mu_l & 0 & 0 & 0 \\
0 & 0 & -\lambda_v & \lambda_v+3\mu_v & -4\mu_v & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -\lambda_v & 4\mu_v & 0 & 0 & 0 & 0 & 0 & 0 \\
-\lambda_l & 0 & 0 & 0 & 0 & \lambda_v+\lambda_l+\mu_l & -\mu_v & 0 & -2\mu_l & 0 & 0 \\
0 & -\lambda_l & 0 & 0 & 0 & -\lambda_v & \lambda_v+\mu_v+\mu_l & -2\mu_v & 0 & 0 & 0 \\
0 & 0 & -\lambda_l & 0 & 0 & 0 & -\lambda_v & 2\mu_v+\mu_l & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -\lambda_l & 0 & 0 & 2\mu_l & 0 & 0 \\
-\lambda_h & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_v+\mu_h & -\mu_v \\
0 & -\lambda_h & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\lambda_v & \mu_v+\mu_h \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix} x = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

By running a linear solver in python we get the following result:
$\pi(0,0,0) = 150000000 \div 1891696937 = 0.07929388532915936$
$\pi(1,0,0) = 375000000 \div 1891696937 = 0.1982347133228984$
$\pi(2,0,0) = 468750000 \div 1891696937 = 0.247793391653623$
$\pi(3,0,0) = 390625000 \div 1891696937 = 0.20649449304468584$
$\pi(4,0,0) = 244140625 \div 1891696937 = 0.12905905815292865$
$\pi(0,1,0) = 34560000 \div 1891696937 = 0.018269311179838318$
$\pi(1,1,0) = 86400000 \div 1891696937 = 0.04567327794959579$
$\pi(2,1,0) = 108000000 \div 1891696937 = 0.05709159743699474$
$\pi(0,2,0) = 3981312 \div 1891696937 = 0.002104624647917374$

$\pi(0,0,1) = 8640000 \div 1891696937 = 0.004567327794959579$
$\pi(1,0,1) = 21600000 \div 1891696937 = 0.011418319487398947$

6. Product-Form solution

The equilibrium state probabilities can also be computed using the product-form formula:

$$\begin{aligned} &\pi\big(n_{\text{voice}}, n_{\text{low}}, n_{\text{high}}\big) = \frac{1}{G} \cdot \frac{\rho_{\text{voice}}^{n_{\text{voice}}}}{n_{\text{voice}}!} \cdot \frac{\rho_{\text{low}}^{n_{\text{low}}}}{n_{\text{low}}!} \cdot \frac{\rho_{\text{high}}^{n_{\text{high}}}}{n_{\text{high}}!}, \text{for } \big(n_{\text{voice}}, n_{\text{low}}, n_{\text{high}}\big) \in S \\ &G \coloneqq \sum_{n \in S} \frac{\rho_{\text{voice}}^{n_{\text{voice}}}}{n_{\text{voice}}!} \cdot \frac{\rho_{\text{low}}^{n_{\text{low}}}}{n_{\text{low}}!} \cdot \frac{\rho_{\text{high}}^{n_{\text{high}}}}{n_{\text{high}}!} \\ &\text{with } \rho_{\text{voice}} \coloneqq \lambda_{\text{voice}}\beta_{\text{voice}}, \rho_{\text{low}} \coloneqq \lambda_{\text{low}}\beta_{\text{low}}, \rho_{\text{high}} \coloneqq \lambda_{\text{high}}\beta_{\text{high}} \end{aligned} \tag{4}$$

If we apply the formula to each state we get:
$G = 12.611312913333334$
$\pi(0,0,0) = 0.07929388532915936$
$\pi(1,0,0) = 0.1982347133228984$
$\pi(2,0,0) = 0.247793391653623$
$\pi(3,0,0) = 0.2064944930446858$
$\pi(4,0,0) = 0.12905905815292865$
$\pi(0,1,0) = 0.018269311179838318$
$\pi(1,1,0) = 0.045673277949595796$
$\pi(2,1,0) = 0.05709159743699474$
$\pi(0,2,0) = 0.0021046246479173745$
$\pi(0,0,1) = 0.004567327794959579$
$\pi(1,0,1) = 0.011418319487398949$

Notice that we get the exact same probabilities as in exercise 5 (minus some floating point errors).

7. Blocking Probability

The blocking probability is just the sum of probabilities of states in which a class is not accepted, or one minus the sum of the probabilities of states in which a class is accepted. We will use the former for the blocking probability of voice calls and the latter for the blocking probability of video calls (this way we have to write less states).

If we look at the Marokov-chain we can notice that:
1. The voice calls are blocked in states $\{(4,0,0), (2,1,0), (0,2,0), (1,0,1)\}$
2. The low-resolution video calls are accepted in states $\{(0,0,0), (1,0,0), (2,0,0), (0,1,0)\}$
3. The high-resolution video calls are accepted in states $\{(0,0,0), (1,0,0)\}$

This gives us the following blocking probabilities:
$B_{\text{voice}} = \pi(4,0,0) + \pi(2,1,0) + \pi(0,2,0) + \pi(1,0,1) = 0.1996735997252397$
$B_{\text{low}} = 1 - \pi(0,0,0) - \pi(1,0,0) - \pi(2,0,0) - \pi(0,1,0) = 0.45640869851448085$
$B_{\text{high}} = 1 - \pi(0,0,0) - \pi(1,0,0) = 0.7224714013479422$

8. Kaufman-Roberts Solution

Kaufman-Roberts works by defining $q(c)$ as the probability that c channels are occupied. Then $q(c)$ follows the following recurrence relation:

$$q(c) = \frac{1}{c} \sum_{k=1}^{K} \rho_k b_k q(c - b_k) \tag{5}$$

Which in our case is:

$$q(c) = \frac{1}{c}\big(\rho_{\text{voice}} q(c-1) + \rho_{\text{low}} 2q(c-2) + \rho_{\text{high}} 3q(c-3)\big) \tag{6}$$

Instead of computing $q(c)$ directly lets start with a function g such that $g(0) = 1$ and then using the recurrence formula to compute $g(c)$ for $c = 1, .., 4$. After that we can compute $q(c)$ by normalizing $g(c)$. Then the blocking probability B_i of class i is the sum of $q(c)$ such that $c + b_i > 4$.

$g(0) = 1$
$g(1) = 30 \cdot \frac{1}{12} q(0) = 2.5$
$g(2) = \frac{1}{2}\big(30 \cdot \frac{1}{12} q(1) + \frac{96}{125} \cdot \frac{3}{10} \cdot 2q(0)\big) = 3.3554$
$g(3) = \frac{1}{3}\big(30 \cdot \frac{1}{12} q(2) + \frac{96}{125} \cdot \frac{3}{10} \cdot 2q(1) + \frac{24}{125} \cdot \frac{3}{10} \cdot 3q(0)\big) = 3.2377666666666673$
$g(4) = \frac{1}{4}\big(30 \cdot \frac{1}{12} q(3) + \frac{96}{125} \cdot \frac{3}{10} \cdot 2q(2) + \frac{24}{125} \cdot \frac{3}{10} \cdot 3q(1)\big) = 2.518146246666667$
$G = q(0) + g(1) + g(2) + g(3) + g(4) = 12.611312913333334$
$q(0) = \frac{g(0)}{G} = 0.07929388532915936$
$q(1) = \frac{g(1)}{G} = 0.19823471332289838$
$q(2) = \frac{g(2)}{G} = 0.2660627028334613$
$q(3) = \frac{g(3)}{G} = 0.2567350987892412$
$q(4) = \frac{g(4)}{G} = 0.1996735997252397$
$B_{\text{voice}} = q(4) = 0.1996735997252397$
$B_{\text{low}} = q(3) + q(4) = 0.45640869851448096$
$B_{\text{high}} = q(2) + q(3) + q(4) = 0.7224714013479422$

Notice that we get the exact same probabilities as in exercise 7 (minus some floating point errors).

9. Kaufman-Roberts Recursion in Python

```py
def kaufman(C: int, p: list[float], b: list[int]) -> list[float]:
    K = len(p)
    q = [0.0] * (C + 1)
    q[0] = 1.0

    for c in range(1, C + 1):
        s = 0.0
        for k in range(K):
            if b[k] <= c:
                s += p[k] * b[k] * q[c - b[k]]
        q[c] = s / c

    G = sum(q)
    q = [qi / G for qi in q]

    B = [0.0] * K
    for k in range(K):
        s = 0.0
        for c in range(C + 1 - b[k], C + 1):
            s += q[c]
        B[k] = s

    return B
```

Listing 1: Kaufman-Roberts Recursion in Python

If we look at the Listing 1, we see that it is doing the same procedure as in the previous exercise. We create an array q and we start with q[0] = 1. With dynamic programming we populate the whole q array using the recurrence formula and the previously computed values. Then we normalize and we compute the blocking probabilities of each class by summing all the q[c] such that c is not enough capacity to accept a new call for that class.

```py
def main():
    C = 4
    lambdas = [30, 96/125, 24/125]
    beta = [1/12, 3/10, 3/10]
    p = [l * b for (l, b) in zip(lambdas, beta)]
    b = [1, 2, 3]
    B = kaufman(C, p, b)
    # Bv = 0.1996735997252397, Bl = 0.45640869851448096, Bh = 0.7224714013479422
    print(f"Bv = {B[0]}, Bl = {B[1]}, Bh = {B[2]}")

if __name__ == "__main__":
    main()
```

Listing 2: Kaufman-Roberts Recursion main

After running Listing 2 we get:
$B_{\text{voice}} = 0.1996735997252397$
$B_{\text{low}} = 0.45640869851448096$
$B_{\text{high}} = 0.7224714013479422$
Which are the exact same probabilities as in exercise 7 and 8 (minus some floating point errors).

II. Optimal distribution of channels over neighboring cells in mobile voice networks

10. Call attempt probabilities

The probability p_i that a call takes place in cell i is:

$$p_i = \frac{\lambda_i}{\sum_{j=1}^{5} \lambda_j} \tag{7}$$

with:
$\lambda_1 = 2$
$\lambda_2 = 5$
$\lambda_3 = 8$
$\lambda_4 = 9$
$\lambda_5 = 11$
which gives us:
$p_1 = \frac{2}{35}$
$p_2 = \frac{5}{35}$
$p_3 = \frac{8}{35}$
$p_4 = \frac{9}{35}$
$p_5 = \frac{11}{35}$

11. Optimal distribution of channels

```py
def erlang(p: list[float], N: int) -> float:
    B = 1.0
    for n in range(1, N + 1):
        B = (p * B) / (n + p * B)
    return B

def erlang_table(C: int, p: list[float]) -> list[list[float]]:
    table = []
    for pi in p:
        row = [erlang(pi, N) for N in range(C + 1)]
        table.append(row)
    return table

def werlang(weights: list[float], n: list[int], table: list[list[float]]) \
    -> float:
    return sum(weights[i] * table[i][ni] for (i, ni) in enumerate(n))
```

Listing 3: Weighted erlang

In Listing 3 we use three different functions to compute the overall blocking probability. The first one is a special case of the Kaufman-Roberts recursion formula, where there is exactly one class and that class occupies one channel. We will derive that in Equation 10. The second function populates a table, where $\text{table}[\text{cell}][\text{c}] = B(\rho_{\text{cell}}, \text{c})$, of all the possible blocking probabilities. This is done to reduce the runtime complexity of the program since the overall blocking probability has to be calculated a lot of times. The third and final function in this code block is the werlang function short

for weighted erlang function. This function calculates the overall blocking probaiblity by using the following formula:

$$\sum_{i=1}^{5} B((\rho_i, n_i) \mid \text{call happend in cell}_i) = \sum_{i=1}^{5} p_i B(\rho_i, n_i) \tag{8}$$

We are yet to derive the formula used in the first function. To do that start by using the Kaufman-Roberts recursion forumla for 1 class and 1 channel.

$$\begin{aligned} q(n) &= \frac{\rho}{n} q(n-1) \\ S_n &:= \sum_{i=0}^{n} g(i) \\ q(n) &= \frac{g(n)}{S_n} \end{aligned} \tag{9}$$

Where $g(c)$ is the un-normalized version of q(0) and g(0) = 1. Then:

$$\begin{aligned} B_n = \frac{q(n)}{S_n} &= \frac{\frac{\rho}{n} g(n-1)}{S_{n-1} + \frac{\rho}{n} g(n-1)} \\ &= \frac{\frac{\rho}{n}}{\frac{S_{n-1}}{g(n-1)} + \frac{\rho}{n}} \\ &= \frac{\frac{\rho}{n}}{\frac{1}{B_{n-1}} + \frac{\rho}{n}} \\ &= \frac{\rho B_{n-1}}{n + \rho B_{n-1}} \end{aligned} \tag{10}$$

So if we start with B = 1.0 we can apply the formula n times to compute B_n.

```py
def optimal(C: int, p: list[float], weights: list[float]) \
    -> tuple[float, list[int]]:
```

```py
    table = erlang_table(C, p)

    min_n = []
    min_b = float("inf")
    for n3 in range(C + 1):
        for n1 in range(C + 1 - n3):
            n2 = C - n3 - n1
            for n4 in range(C + 1 - n3):
                n5 = C - n3 - n4
                n = [n1, n2, n3, n4, n5]
                b = werlang(weights, n, table)
                if b < min_b:
                    min_n = n
                    min_b  = b
    return (min_b, min_n)
```

Listing 4: Optimal channels distribution

In Listing 4 we have a function that iters over all the possible distributions of channels and calculates the overall blocking probability. It keeps track of the minimum and returns it. To iterate over all the possible channel distributions we could naively iterate over all possible 5-tuple $n \in Z_C^5$ and check if it is a valid channel distribution given our constraints. But that would be inefficent. Instead what we can do is chose the number of channels for the third cell, now we know that to be a valid channel distribution should have $(n_1 + n_2 \leq C - n_3) \wedge (n_4 + n_5 \leq C - n_3)$, but since we are interested in the minimum it does not make sense to use less channels than what we are given, so we will only consider the case that $(n_1 + n_2 = C - n_3) \wedge (n_4 + n_5 = C - n_3)$. So once we have chosen n_3 we can iter over all the possible n_1 and n_4 and from that constraints calculate n_2 and n_5. Reducing the runtime complexity from $O(C^5)$ to $O(C^3)$.

```py
def main():
    C = 32
    lambdas = [2, 5, 8, 9, 11]
    beta = [1.5, 1.5, 1.5, 1.5, 1.5]
    p = [l * b for (l, b) in zip(lambdas, beta)]
    lsum = sum(lambdas)
    weights = [l / lsum for l in lambdas]

    B, n = optimal(C, p, weights)
    # C = 32, B = 0.27431686232726826, n = [8, 15, 9, 10, 13]
    print(f"C = {C}, B = {B}, n = {n}")

```

```py
if __name__ == "__main__":
    main()
```

Listing 5: Optimal channels distribution main

Listing 5 gives us a overall blocking probaiblity of $0.27431686232726826 \approx 27.4\%$, distributing this ammount of channels per cell:
$n_1 = 8$
$n_2 = 15$
$n_3 = 9$
$n_4 = 10$
$n_5 = 13$

12. Optimal distribution below 1%

For this problem we can reuse the function from the previous exercise and keep increasing the capacity until we get a blocking probability less than 1%. A more optimal approach to further improve performance of the algorithm would be to use binary search, but the code run already fast enough so we keept the more simple solution.

```py
def main():
    C = 32
    lambdas = [2, 5, 8, 9, 11]
    beta = [1.5, 1.5, 1.5, 1.5, 1.5]
    p = [l * b for (l, b) in zip(lambdas, beta)]
    lsum = sum(lambdas)
    weights = [l / lsum for l in lambdas]

```

```py
    while B >= 0.01:
        C += 1
        B, n = optimal(C, p, weights)
    # C = 66, B = 0.009938133851637975, n = [18, 28, 20, 21, 25]
    print(f"C = {C}, B = {B}, n = {n}")

if __name__ == "__main__":
    main()
```

Listing 6: Optimal channels distribution below 1%

Listing 6 gives us a overall blocking probaiblity of bearly less than 1%, with a capacity of 66 and distributing this ammount of channels per cell:

$n_1 = 18$
$n_2 = 28$
$n_3 = 20$
$n_4 = 21$
$n_5 = 25$

III. Traffic Management in IP networks

Before shaping/policing we have this incoming traffic.

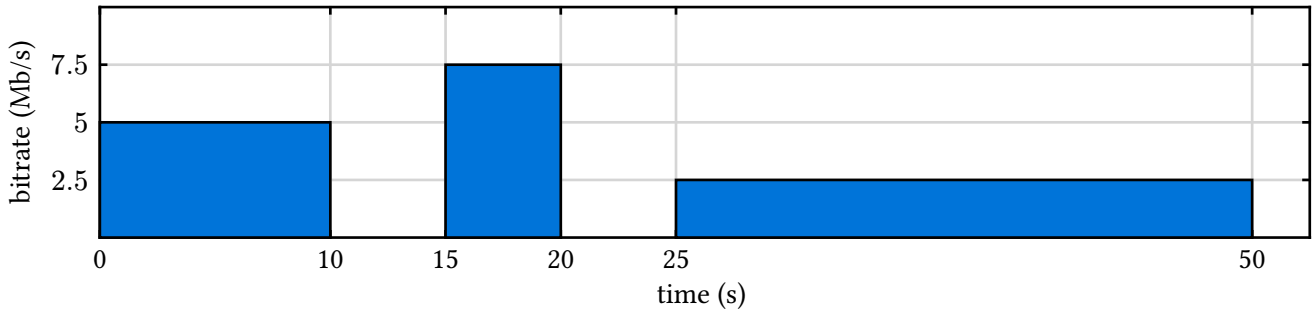


Figure 1: Bitrate before shaping/policing

13. Traffic Shaping Delay Graph

1. For the first burst we have a peak rate of 5Mb/s distributed over 10 seconds. Since the shaping rate is 4Mb/s the shaping buffer level rises at rate $5\text{Mb/s} - 4\text{Mb/s} = 1\text{Mb/s}$. Which means that at the end of the burst the buffer level is $1\text{Mb/s} \cdot 10\text{s} = 10\text{Mb}$ and the delay is $\frac{10\text{Mb}}{4\text{Mb/s}} = 2.5\text{s}$.
2. For the first burst we have a peak rate of 7.5Mb/s distributed over 5 seconds. Since the shaping rate is 4Mb/s the shaping buffer level rises at rate $7.5\text{Mb/s} - 4\text{Mb/s} = 3.5\text{Mb/s}$. Which means that at the end of the burst the buffer level is $3.5\text{Mb/s} \cdot 5\text{s} = 17.5\text{Mb}$ and the delay is $\frac{17.5\text{Mb}}{4\text{Mb/s}} = 4.375\text{s}$.
3. For the third and final burst we have a peak rate of 2.5Mb/s, which is less than the 4Mb/s shaping rate. This means that the third burst is not delayed.

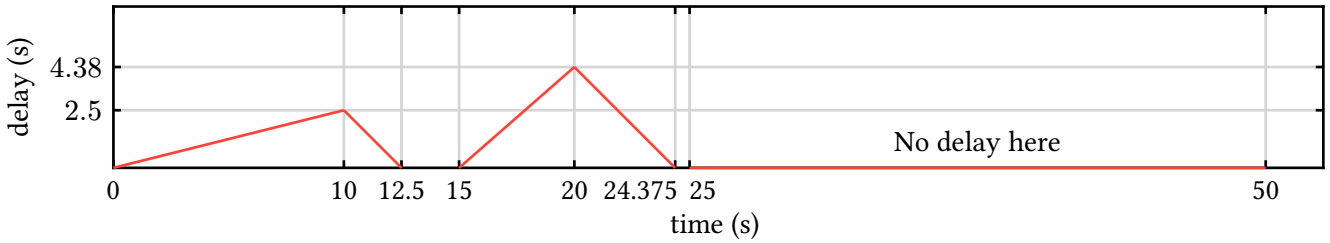


Figure 2: Conforming traffic after policing

14. Traffic Shaping Bitrate Graph

1. For the first burst we have a peak rate of 5Mb/s distributed over 10 seconds. Which means that the total ammount of traffic is $5\text{Mb/s} \cdot 10\text{s} = 50\text{Mb}$. Since the shaping rate is 4Mb/s the shaped burst is going to be 4Mb/s distributed over $\frac{50\text{Mb}}{4\text{Mb/s}} = 12.5\text{s}$.
2. For the second burst we have a peak rate of 7.5Mb/s distributed over 5 seconds. Which means that the total ammount of traffic is $7.5\text{Mb/s} \cdot 5\text{s} = 37.5\text{Mb}$. Since the shaping rate is 4Mb/s the shaped burst is going to be 4Mb/s distributed over $\frac{37.5\text{Mb}}{4\text{Mb/s}} = 9.375\text{s}$.
3. For the third and final burst we have a peak rate of 2.5Mb/s, which is less than the 4Mb/s shaping rate. This means that the third burst remains unchanged.

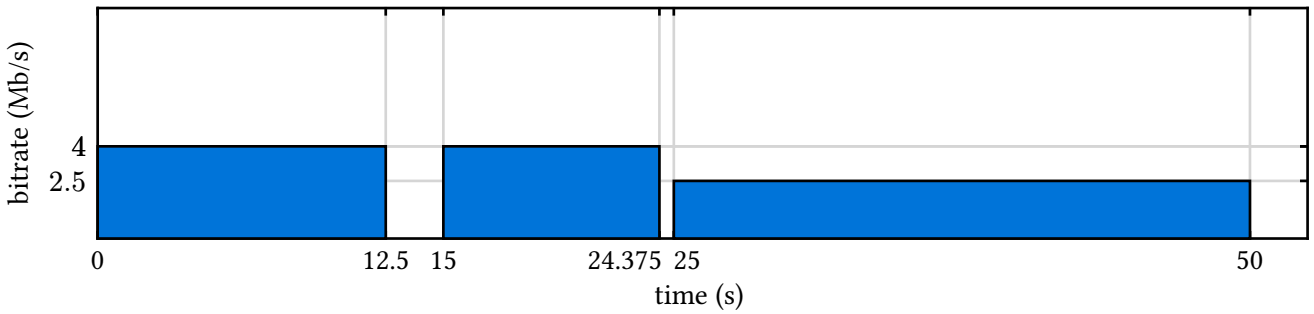


Figure 3: Bitrate after shaping

15. Traffic Policing Bitrate Graph

Trick question the bitrate is the same as before. Policing with marking (not when we drop the packets) does not reduce the bitrate, it only signals which packets exceed the allowed rate.

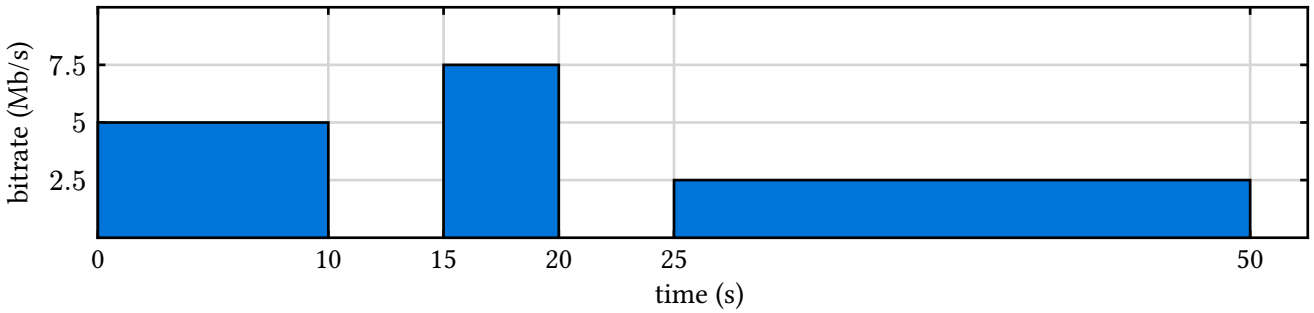


Figure 4: Bitrate before/after policing

16. Traffic Policing Conforming Graph

Note that the "water volume" takes $\frac{8\text{Mb}}{4\text{Mb/s}} = 2s$ to become empty once it has reached the burst tolerance. Since we have a 5 second spacing between each burst we can consider each burst separately.

1. For the first burst we have a peak rate of 5Mb/s distributed over 10 seconds. Since the leak rate is 4Mb/s and the burst tolerance is 8Mb the "water volume" rises at rate $5\text{Mb/s} - 4\text{Mb/s} = 1\text{Mb/s}$, and after $\frac{8\text{Mb}}{1\text{Mb/s}} = 8\text{s}$ the "water volume" reaches the burst tolerance. Which means that in the first 8 seconds (in blue) the traffic is marked as conforming and between 8 and 10 seconds (in red) only $\frac{4}{5}$ of the packets are marked as conforming, and the remaining $\frac{1}{5}$ non-conforming.
2. For the second burst we have a peak rate of 7.5Mb/s distributed over 5 seconds. Since the leak rate is 4Mb/s and the burst tolerance is 8Mb the "water volume" rises at rate $7.5\text{Mb/s} - 4\text{Mb/s} = 3.5\text{Mb/s}$, and after $\frac{8\text{Mb}}{3.5\text{Mb/s}} \approx 2.28\text{s}$ the "water volume" reaches the burst tolerance. Which means that in the first 2.28 seconds (in blue) the traffic is marked as conforming and between 2.28 and 5 seconds (in red) only $\frac{4}{7.5}$ of the packets are marked as conforming, and the remaining $\frac{3.5}{7.5}$ non-conforming.
3. For the third and final burst we have a peak rate of 2.5Mb/s, which is less than the 4Mb/s leak rate. This means that the third burst is all compleately marked as conforming.

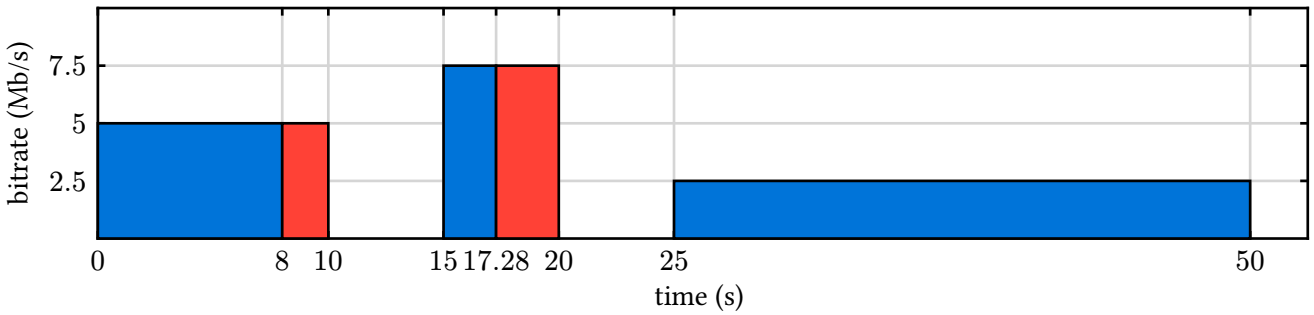


Figure 5: Conforming traffic after policing

IV. Performance of TCP-based networks

17. Downside of TCP Slow Start

The main downside of TCP Slow Start is that it can significantly underutilize available bandwidth at the beginning of a connection. Because TCP does not know the amount of available bandwidth at startup, it begins with a small congestion window that increases exponentially. This requires multiple RTTs to ramp up, which can lead to noticeable performance degradation, as the network is not used at full capacity during this phase.

This issue is especially apparent when transferring many small files, by the time the congestion window grows large enough to fully utilize the bandwidth, the file transfer is already complete. For example, this was a known problem in early versions of HTTP, where many small files were commonly transferred.

18. Slow Start Transfer Times

15KB

The client first establishes a TCP connection using the three-way handshake, which takes one RTT (60 ms). After the connection is established, the client transmits a single request packet to the server. This packet requires half RTT to reach the server. The server then processes the request for 50 ms and immediately begins transmitting the first response packet, which arrives at the client after the remaining half RTT. Up to this point, the transfer time is 170 ms (60ms + 50ms + 60ms).

During Slow Start, the server's congestion window doubles every RTT. After one RTT, the client receives two packets. After another RTT, the window doubles again and the client receives four packets. After one additional RTT, the window doubles to eight packets, although the server only needs to send 5.140 bytes of data, corresponding to four packets. At that point, the transfer completes.

The total time required is therefore:

- 1 RTT for the TCP handshake (60 ms)
- 50 ms of server processing
- 4 RTTs during Slow Start (4 x 60 ms)

This results in a total transfer time of 350 ms.

25KB and 40KB

The procedure is identical to the 15KB case, except that transferring these larger files requires one additional congestion window doubling. This adds one more RTT to the Slow Start phase. Therefore, the total transfer time for both the 25KB and 40KB files is 410 ms.

Final Results

The results can be summarized in the following table.

File size	Number of packets	Slow Start RTTs	Transmission Time
15KB	11	4	350ms
25KB	18	5	410ms
45KB	29	5	410ms