Analysis of buffer system capacity's relation with number of IRIDIUM® satellite network's servers

Paras Balani(2023B4A70738H)
Aditya Sinha(2023B4A40634H)
Rishabh Singh(2023B4A70721H)

Birla Institute of Technology and Sciences, Pilani, Hyderabad Campus.

Acknowledgment

We would like to express our sincere gratitude to **Major Carl E. Fossa** , **Dr. Richard A. Raines** , **Lt. Col. Gregg H. Gunsch**, and **Dr. Michael A. Temple** for their valuable research contributions [1].Their work has been instrumental in shaping the foundation of this research, and we have extensively referenced their findings in this paper.

We are profoundly indebted to **PTV Praveen Kumar Sir** for his outstanding pedagogy in **Operational Research**. His unique teaching methodology, which combines rigorous theoretical foundations with practical applications, has been transformative in our understanding of the subject.

Academic Integrity and Copyright Notice

This document represents original academic work and is shared solely for the purpose of scholarly review and constructive feedback. It may form the basis of future research or publication. The contents—including text, figures, and ideas—are the intellectual property of the author(s) and are protected under applicable copyright and academic integrity standards. Any reproduction, distribution, citation, or use of this material without prior written consent from the author(s) is strictly prohibited. Please respect the spirit of academic collaboration by refraining from unauthorized use.

Abstract

This paper investigates the mathematical relationship between buffer system capacity (threshold m) and the number of servers (s) in the context of the IRIDIUM® Low Earth Orbit (LEO) satellite network, utilizing models from queuing theory. Focusing specifically on the $(M/M/s)$: $(FCFS/m/\infty)$ queuing system, the study aims to determine the minimum buffer size required such that the system's performance closely approximates that of an infinite-capacity queue, a critical consideration for resource-constrained satellite systems.

The research first provides a foundational overview of queuing theory, introducing essential notations, performance metrics, and model assumptions commonly applied in telecommunications and space networking. It then delves into the operational specifics of the IRIDIUM® satellite constellation, detailing its hybrid TDMA/FDMA access scheme, spot beam technology, inter-satellite links, and network-level traffic characteristics. Realistic network parameters—such as packet arrival rates, service rates, and traffic intensities—are derived based on published IRIDIUM® technical data.

Analytically, the study presents complete derivations for steady-state probabilities, queue lengths, waiting times, and blocking probabilities for both infinite and finite buffer systems. The relationship between buffer threshold m and the number of servers s is investigated via numerical modeling using Python. By plotting key system metrics (queue length, waiting time, system occupancy) against increasing buffer sizes for varying numbers of servers, the research identifies the convergence point (m_{th}) where system performance is indistinguishable from that of an ideal infinite-buffer system.

Results show a pronounced non-linear reduction in required buffer capacity with the initial addition of servers in small systems, transitioning to a near-linear relationship ($m \approx s$) in larger constellations. These findings are further contextualized with insights from modern reinforcement learning-based network optimization, emphasizing the practical design implications for satellite mega-constellations.

The work concludes with actionable recommendations for buffer sizing in satellite communication design, balancing the trade-offs between resource expenditure, packet loss (blocking probability), and service latency. The methodology and code provided are broadly applicable to other multi-server, limited-buffer queuing systems found in both terrestrial and non-terrestrial networks.

1 Introduction to Queuing Theory

Queuing theory is a mathematical study of waiting lines or queues. It provides a framework for analyzing systems in which customers—broadly defined as people, data packets, machines, or tasks—arrive, wait for service if necessary, receive service, and then depart. A queuing system typically consists of a population of prospective customers, an arrival mechanism, a queue or buffer, a service mechanism, and a defined service discipline that governs the order in which customers are served.

At its core, a basic queuing system involves customers arriving from a source population, entering a queue if the server is busy, receiving service, and then leaving the system. The population can be finite or infinite depending on whether the arrival rate is influenced by the number of customers in the system. An infinite population is commonly assumed in theoretical models for simplicity when the arrival rate remains constant regardless of the system state.

The *arrival pattern* is usually modeled as a Poisson process, under the assumption that the *inter-arrival times* are exponentially distributed and independent. This assumption simplifies the mathematical analysis and reflects many real-world random arrival processes. The *service time* distribution is also often modeled as exponential and independent for different customers, forming the basis for the Markovian models such as the M/M/1 or M/M/s queues.

The *queue discipline*, which determines the rule for selecting the next customer for service, significantly influences system performance. Common disciplines include *First Come First Serve (FCFS)*, *Last Come First Serve (LCFS)*, *Service in Random Order (SIRO)*, and *priority-based schemes*.

The *state of the system* at any time is defined by the total number of customers present, including both those in service and those waiting in the queue. Understanding this state distribution is essential for calculating important performance measures such as the average queue length, average waiting time, and server utilization.

Queuing theory is not merely theoretical; it has widespread practical applications across multiple domains. For instance, in hospital pharmacy optimization, queuing models have been used to simulate and improve operational efficiency by reducing patient wait times and balancing staff workloads [9]. Similarly, in the banking sector, the application of queuing theory has enabled better teller scheduling, enhancing customer satisfaction and reducing service bottlenecks [10].

In transportation systems, queuing theory supports the optimization of limited infrastructure, such as parking space allocation in Bus Rapid Transit (BRT) systems [11], and the efficiency of direct rolling processes in manufacturing [27]. Its ability to model congestion and stochastic delays also informs strategies for urban traffic management [29] and energy-efficient logistics in green supply chains [8].

Further, researchers have extended queuing theory into more advanced domains. For example, [13] explores the design of queuing-based simulations for strategic economic experiments, while [14] develops Markov queuing models for hospital bed resource allocation. Queuing theory also aids in decision-making for facility layout and capacity planning, as shown in the optimization of electric vehicle charging stations [26] and the simulation of airport check-in counters using both classical and fuzzy queuing models [25].

Beyond classical models, advancements include the use of *spectral expansion methods* for systems with shifted Erlang and hyper-Erlang distributions [23], and random graph-based approaches for linking queuing with reliability analysis [28]. These developments push the boundaries of queuing theory, making it more adaptable to real-world complexities and hybrid environments.

In telecommunication and networking, queuing models have been employed to bridge performance gaps in routers and switches [16], and to analyze response times under preemptive priority scheduling with switching overhead [24]. Additionally, queuing theory has guided the design of resilient satellite communication systems [8] and mobile service frameworks [12].

Thus, queuing theory emerges as a robust analytical tool that provides both *mathematical rigor* and *practical insights* for system performance evaluation and optimization. Its continued evolution—through integration with computer simulation, optimization methods, and interdisciplinary frameworks—ensures its relevance in addressing contemporary challenges in service operations, logistics, healthcare, telecommunications, and beyond.

2 Terminology and Notations in Queuing Theory

In the analysis of queuing systems, we introduce several notations and definitions to describe the stochastic behavior of the system. Let N denote the number of customers present in the system at any given instant, which includes both customers waiting in the queue and those currently being served. A related variable, Q, represents the number of customers specifically in the queue at any instant. Both N and Q are treated as discrete random variables with respective ranges $R_N = R_Q = \{0, 1, 2, \dots\}$.

We define the probability that the system contains exactly n customers at a particular time T as $P_n(T) = \mathbb{P}(N(T) = n)$, which can also be interpreted as the probability that there are n customers in the system at time T. In the case where time-dependence is not explicitly considered, we write this as $P_n = \mathbb{P}(N = n)$, where $n \in \{0, 1, 2, \dots\}$.

The parameter s denotes the number of servers operating in parallel within the queuing system. The arrival and service dynamics are characterized by the state-dependent arrival and service rates. Specifically, let λ_n be the mean arrival rate when the system is in state n, i.e., when there are n customers in the system. Similarly, let μ_n denote the mean service rate under the same condition. It is important to distinguish μ_n from the mean service time, as the two are reciprocally related. The mean service time is given by the inverse of the mean service rate, i.e.,

$$\text{Mean service time} = \frac{1}{\text{Mean service rate}}.$$

This foundational terminology and these notations are critical for the formulation and analysis of queuing models, particularly in the study of birth-death processes, Markovian queues, and performance evaluation metrics such as average queue length, waiting time, and system utilization.

3 Expected Number of Customers in the System and the Queue

Let L denote the expected number of customers in the entire system at any arbitrary point in time. Mathematically, this is represented by the expected value of the random variable N, which counts the number of customers in the system. The expression for L is given by:

$$L = \mathbb{E}[N] = \sum_{n=0}^{\infty} nP_n,$$

where P_n denotes the probability that there are exactly n customers in the system.

Similarly, let L_q denote the expected number of customers specifically in the queue at any given instant. This is the expectation of the random variable Q, which counts the number of customers waiting (i.e., not being served). We write:

$$L_q = \mathbb{E}[Q].$$

To compute $\mathbb{E}[Q]$ analytically, we begin with:

$$\mathbb{E}[Q] = \sum_{k=0}^{\infty} k\mathbb{P}(Q = k).$$

If there are s servers, and assuming k customers are in the queue, the total number of customers in the system must be $k + s$. Thus, we reparameterize the summation:

$$= \sum_{k=0}^{\infty} k\mathbb{P}(N = k + s).$$

Setting $n = k + s$, we can rewrite the summation in terms of n:

$$= \sum_{n=s}^{\infty} (n - s)\mathbb{P}(N = n) = \sum_{n=s}^{\infty} (n - s)P_n.$$

4 Server Utilization and Waiting Time

Server utilization, denoted by ρ, is a key performance metric that measures the fraction of time the server(s) are busy. For a system with s parallel servers, the utilization is given by:

$$\rho = \frac{\lambda}{s\mu},$$

where λ is the mean arrival rate and μ is the mean service rate of a single server. In the special case of a single-server system ($s = 1$), this simplifies to:

$$\rho = \frac{\lambda}{\mu}.$$

Hence, the server utilization ρ and the fraction of time the server is busy are numerically equal in a single-server setting.

Let W denote the mean time a customer spends in the system (both in the queue and being served), and let W_q denote the mean waiting time a customer spends specifically in the queue. There exists a fundamental relationship between these two quantities:

$$W = W_q + \frac{1}{\mu},$$

assuming that the mean service time is $1/\mu$. This expression states that the total expected time in the system is the sum of the expected time waiting in the queue and the expected time spent in service.

These relationships are central to queuing theory, particularly in the analysis of M/M/1 and M/M/s models, and form the basis for performance evaluation and system optimization.

5 Little's Law

In classical queueing theory, **Little's Law** provides a fundamental relationship that links the average number of customers in a system, L, to the average arrival rate, λ, and the average time a customer spends in the system, W. Mathematically, this is expressed as:

$$L = \lambda W$$

Similarly, the average number of customers in the queue, denoted L_q, relates to the average waiting time in the queue, W_q, through the expression:

$$L_q = \lambda W_q$$

These elegant relationships hold true under the assumption of an **infinite capacity system**, where there is no restriction on the number of customers that can enter or remain in the system.

However, in **finite-capacity systems**, these classical forms of Little's Law do not apply directly. When there is a limit on the maximum number of customers that the system can accommodate, the arrival rate into the system is no longer constant. This is because some arriving customers may be blocked or turned away due to capacity restrictions, especially in loss systems or systems with finite buffers.

To accommodate this reality, a **modified version of Little's Law** is used. In this modified framework, the arrival rate is replaced by an **effective arrival rate**, denoted by $\overline{\lambda}$, which accounts for the probability that a customer is successfully admitted into the system. The modified Little's Law for a finite system is therefore given by:

$$L = \overline{\lambda} W$$

$$L_q = \overline{\lambda} W_q$$

Here, $\overline{\lambda}$ represents the average rate at which customers are actually admitted to the system. It is computed as a **weighted sum** of the state-dependent arrival rates λ_n multiplied by the steady-state probabilities P_n of the system being in state n. This is formally written as:

$$\overline{\lambda} = \sum_{n=0}^{\infty} \lambda_n P_n$$

This expression accounts for the possibility that the system may reject new arrivals when it is full, thus providing a realistic measure of throughput. The use of the effective arrival rate ensures that Little's Law remains valid in the context of finite systems by reflecting the actual flow of customers through the system.

6 Kendall's Notation

A queuing system is commonly described using **Kendall's notation**, which has the general form:

$$(a/b/c) : (d/e/f)$$

Each component in this notation represents a specific characteristic of the queuing system:

- a – The distribution of inter-arrival times between successive customers.
- b – The distribution of service times.
- c – The number of parallel servers in the system.
- d – The queue discipline (i.e., the rule used to select the next customer for service, e.g., FIFO).
- e – The capacity of the system (i.e., the maximum number of customers allowed in the system including those in service).
- f – The size of the potential customer population.

A commonly used symbol in Kendall's notation is **M**, which denotes a Markovian (i.e., exponential) distribution. This symbol can be used for both a and b when the arrival and service times follow exponential distributions, respectively.

7 Derivations of useful formulae of (M/M/s):(FCFS/inf/inf) model

7.1 Kendal Notations:

- **Arrival Process:** Poisson with rate λ (i.i.d. exponential interarrival times).
- **Service Process:** Exponential service times with rate μ per server.
- **Servers:** s identical servers.
- **Queue Discipline:** First-Come-First-Served (FCFS).
- **Population:** Infinite.
- **Capacity:** Infinite.

The system is modeled as a birth-death process with states $n = 0, 1, 2, \ldots$, representing the number of jobs in the system.

7.2 Transition Rates

- **Arrival Rate (λ_n):**

$$\lambda_n = \lambda \quad \text{for all } n \geq 0.$$

- **Service Rate (μ_n):**

$$\mu_n = \begin{cases} n\mu & \text{if } 1 \leq n < s, \quad \text{(all jobs served in parallel)} \\ s\mu & \text{if } n \geq s. \quad \text{(all servers busy, queue forms)} \end{cases}$$

7.3 State Transition Equations (Steady-State)

For steady-state probabilities P_n, the balance equations are:

1. **State 0:**

$$\lambda P_0 = \mu P_1.$$

2. **States $1 \leq n < s$:**

$$(\lambda + n\mu)P_n = \lambda P_{n-1} + (n+1)\mu P_{n+1}.$$

3. **States $n \geq s$:**

$$(\lambda + s\mu)P_n = \lambda P_{n-1} + s\mu P_{n+1}.$$

7.4 Solving the Balance Equations

Recursive Solution for P_n

- **For $0 \leq n < s$:** From $\lambda P_0 = \mu P_1$, we get $P_1 = \frac{\lambda}{\mu} P_0$. Similarly, for $n = 1$:

$$(\lambda + \mu)P_1 = \lambda P_0 + 2\mu P_2 \implies P_2 = \frac{\lambda^2}{2\mu^2} P_0.$$

Generalizing:

$$P_n = \frac{\lambda^n}{n!\mu^n} P_0 = \frac{\rho^n}{n!} P_0, \quad \text{where } \rho = \frac{\lambda}{\mu}.$$

- **For $n \geq s$:** The system reduces to a repeating pattern (all servers busy):

$$P_{n+1} = \frac{\lambda}{s\mu} P_n = \rho_s P_n, \quad \text{where } \rho_s = \frac{\lambda}{s\mu}.$$

Thus:

$$P_n = \frac{\lambda^n}{s!\mu^n s^{n-s}} P_0 = \frac{\rho^n}{s! s^{n-s}} P_0.$$

7.5 Normalization Condition

The probabilities must sum to 1:

$$\sum_{n=0}^{\infty} P_n = P_0 \left(\sum_{n=0}^{s-1} \frac{\rho^n}{n!} + \sum_{n=s}^{\infty} \frac{\rho^n}{s! s^{n-s}} \right) = 1.$$

The second sum is a geometric series for $n \geq s$:

$$\sum_{n=s}^{\infty} \frac{\rho^n}{s! s^{n-s}} = \frac{\rho^s}{s!} \sum_{k=0}^{\infty} \left(\frac{\rho}{s}\right)^k = \frac{\rho^s}{s!} \cdot \frac{1}{1-\rho_s}, \quad \text{if } \rho_s < 1.$$

Thus:

$$P_0 = \left[\sum_{n=0}^{s-1} \frac{\rho^n}{n!} + \frac{\rho^s}{s!(1-\rho_s)} \right]^{-1}.$$

7.6 Performance Metrics

7.6.1 Queue Length L_q

Jobs waiting in the queue (excluding those in service):

$$L_q = \sum_{n=s}^{\infty} (n-s) P_n = P_0 \frac{\rho^s \rho_s}{s!(1-\rho_s)^2}.$$

7.6.2 Waiting Time (W_q)

By Little's Law:

$$W_q = \frac{L_q}{\lambda}.$$

7.6.3 Total Time in System (W)

$$W = W_q + \frac{1}{\mu}.$$

7.6.4 Total Jobs in System (L)

$$L = \lambda W = L_q + \frac{\lambda}{\mu}.$$

7.7 Convergence Condition

For steady-state to exist, the system must be stable:

$$\rho_s = \frac{\lambda}{s\mu} < 1 \quad \text{(i.e., arrival rate < maximum service rate)}.$$

8 Derivations of Useful Formulas of (M/M/s): (FCFS/m/inf) Queuing Model

8.1 Kendal Notations:

- **Arrival Process:** Poisson with rate λ (i.i.d. exponential interarrival times).
- **Service Process:** Exponential service times with rate μ per server.
- **Servers:** s identical servers.
- **Queue Discipline:** First-Come-First-Served (FCFS).
- **Population:** Infinite.
- **Capacity:** m.

8.2 Transition Rates

Arrival rate (λ_n):

$$\lambda_n = \begin{cases} \lambda, & 0 \leq n \leq m-1 \\ 0, & n \geq m \end{cases}$$

Service rate (μ_n):

$$\mu_n = \begin{cases} n\mu, & 1 \leq n \leq s \\ s\mu, & s \leq n \leq m \end{cases}$$

8.3 Steady-State Balance Equations

Let P_n be the steady-state probability of having n customers in the system. Define $\rho = \frac{\lambda}{\mu}$.

8.3.1 For $n = 0$

$$\lambda P_0 = \mu P_1 \Rightarrow P_1 = \frac{\lambda}{\mu} P_0 = \rho P_0$$

8.3.2 For $1 \leq n \leq s-1$

$$\lambda P_{n-1} = n\mu P_n \Rightarrow P_n = \frac{\lambda}{n\mu} P_{n-1} = \frac{\rho^n}{n!} P_0$$

8.3.3 For $s \leq n \leq m$

$$\lambda P_{n-1} = s\mu P_n \Rightarrow P_n = \frac{\lambda}{s\mu} P_{n-1} = \frac{\rho^n}{s! s^{n-s}} P_0$$

8.3.4 For $n > m$

$$P_n = 0$$

8.4 Normalization Condition

$$\sum_{n=0}^{m} P_n = 1$$

$$P_0 \left[1 + \sum_{n=1}^{s-1} \frac{\rho^n}{n!} + \sum_{n=s}^{m} \frac{\rho^n}{s! s^{n-s}} \right] = 1$$

$$P_0 = \left[\sum_{n=0}^{s-1} \frac{\rho^n}{n!} + \frac{\rho^s}{s!} \sum_{k=0}^{m-s} \left(\frac{\rho}{s}\right)^k\right]^{-1}$$

Special Cases:

- If $\frac{\rho}{s} \neq 1$:
$$\sum_{k=0}^{m-s} \left(\frac{\rho}{s}\right)^k = \frac{1-\left(\frac{\rho}{s}\right)^{m-s+1}}{1-\frac{\rho}{s}}$$

- If $\frac{\rho}{s} = 1$:
$$\sum_{k=0}^{m-s} \left(\frac{\rho}{s}\right)^k = m-s+1$$

8.5 Performance Measures

8.5.1 Expected Queue Length L_q

$$L_q = \sum_{n=s}^{m}(n-s)P_n = \sum_{k=1}^{m-s} kP_{s+k}$$

$$P_{s+k} = \frac{\rho^{s+k}}{s!s^k}P_0 \Rightarrow L_q = \frac{\rho^s}{s!}P_0 \sum_{k=1}^{m-s} k\left(\frac{\rho}{s}\right)^k$$

Use the identity:

$$\sum_{k=1}^{N} kx^k = \frac{x\left[1-(N+1)x^N + Nx^{N+1}\right]}{(1-x)^2}, \quad x \neq 1$$

$$L_q = \frac{\rho^{s+1}}{s!s}P_0\left[\frac{1-(m-s+1)(\rho/s)^{m-s} + (m-s)(\rho/s)^{m-s+1}}{(1-\rho/s)^2}\right]$$

8.5.2 Effective Arrival Rate ($\bar{\lambda}$)

$$\bar{\lambda} = \lambda(1-P_m)$$

8.5.3 Waiting Times

Using Little's Law:

$$W_q = \frac{L_q}{\bar{\lambda}}, \quad W = W_q + \frac{1}{\mu}$$

8.5.4 Expected Number in System (L)

$$L = \sum_{n=0}^{m} nP_n = \sum_{n=1}^{s-1} nP_n + \sum_{n=s}^{m} nP_n$$

$$= \sum_{n=1}^{s-1} \frac{\rho^n}{(n-1)!}P_0 + s\sum_{n=s}^{m} P_n + L_q \Rightarrow L = \rho(1-P_m) + L_q$$

8.6 Summary of Key Formulae

8.6.1 Steady-State Probabilities

$$P_n = \begin{cases} \frac{\rho^n}{n!}P_0, & 0 \le n \le s-1 \\ \frac{\rho^n}{s!s^{n-s}}P_0, & s \le n \le m \\ 0, & n > m \end{cases}$$

$$P_0 = \left[\sum_{n=0}^{s-1} \frac{\rho^n}{n!} + \frac{\rho^s}{s!}\left(\frac{1-(\rho/s)^{m-s+1}}{1-\rho/s}\right)\right]^{-1}, \quad \text{if } \rho/s \neq 1$$

8.6.2 Queue Length (L_q)

$$L_q = \frac{\rho^{s+1}}{s!s}P_0\left[\frac{1-(m-s+1)(\rho/s)^{m-s} + (m-s)(\rho/s)^{m-s+1}}{(1-\rho/s)^2}\right]$$

8.6.3 Effective Arrival Rate

$$\bar{\lambda} = \lambda(1-P_m)$$

8.6.4 Waiting Times

$$W_q = \frac{L_q}{\bar{\lambda}}, \quad W = W_q + \frac{1}{\mu}$$

8.6.5 Expected Number in System

$$L = \rho(1-P_m) + L_q$$

9 Data of Arrival rate and Service rate from IRIDIUM® satellite network:-

The following data is inspired from citations [1] to [7]

9.1 Personal Communication Services (PCS) and LEO Satellite Systems

Personal Communication Services (PCS) encompass a broad spectrum of wireless communication technologies designed to enable continuous connectivity across various platforms, including mobile voice and data services, text messaging, and internet access. These systems provide users with reliable communication capabilities regardless of location, with common implementations including modern cellular networks such as 4G LTE and 5G New Radio, as well as satellite phone systems.

A significant development in enhancing PCS capabilities involves the utilization of Low Earth Orbit (LEO) satellite networks. These satellites operate at altitudes between 300 and 2,000 kilometers, substantially closer to Earth's surface compared to traditional geostationary satellites that orbit at over 35,000 kilometers. The reduced orbital distance offers two primary technical advantages: significantly lower transmission latency, which improves real-time communication performance, and enhanced coverage capabilities, particularly beneficial for rural and geographically isolated areas. However, the operational characteristics of LEO satellites present unique challenges—their high orbital velocity, approximately 27,000 kilometers per hour, requires the deployment of numerous satellites in coordinated constellations to ensure uninterrupted service coverage. This innovative approach to satellite network design represents a major advancement in telecommunications infrastructure, offering improved performance characteristics while addressing the growing demand for universal connectivity.

9.2 IRIDIUM® Satellite Network

IRIDIUM® holds the distinction of being the first commercial Low Earth Orbit (LEO) satellite network to incorporate inter-satellite links (ISLs). These ISLs enable direct communication between satellites, forming a space-based mesh network that minimizes reliance on ground infrastructure.

The IRIDIUM® constellation consists of 66 active satellites distributed across six orbital planes, with each plane containing 11 satellites. Orbiting at an altitude of approximately 780 km (485 miles), these satellites provide global coverage. Each satellite employs 48 focused spot beams, functioning similarly to cellular towers in space to direct connectivity to specific regions.

9.3 Technical Analysis of Iridium Satellite Communication System

9.3.1 Access Scheme: Hybrid TDMA/FDMA

The Iridium system employs a sophisticated **hybrid TDMA/FDMA** (Time Division Multiple Access/Frequency Division Multiple Access) scheme to optimize spectrum utilization:

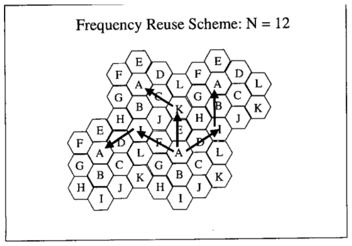
- **TDMA Component**:
  - Divides the transmission timeline into discrete time slots (90 ms each)
  - Each user is allocated specific time intervals for transmission
  - Operates similarly to taking turns in a game where each player gets equal opportunity
  - Mathematical representation: $T_{frame} = N_{slots} \times T_{slot}$
- **FDMA Component**:
  - Divides the available frequency spectrum into 41.67 kHz channels
  - Each beam operates on distinct frequency channels
  - Analogous to different radio stations operating on separate frequencies
  - Frequency allocation: $f_{total} = \sum_{n=1}^{N} f_{channel_n}$
- **Hybrid Operation**:
  - Combines both time and frequency division for maximum efficiency
  - Creates a two-dimensional resource grid (time $\times$ frequency)
  - Allows simultaneous support for multiple users without interference

9.3.2 Frequency Reuse Pattern

The system implements a sophisticated 12-cell frequency reuse pattern to maximize spectral efficiency:

$$N = I^2 + I \cdot J + J^2 \quad \text{where } (I = 2, J = 2) \tag{1}$$

- $N = 12$ represents the number of unique frequency groups
- I and J are integer steps that determine the spatial separation between cells using the same frequencies
- The pattern forms a hexagonal (honeycomb) structure in space
- Ensures sufficient distance between cells using identical frequencies to prevent co-channel interference



credit: dl.acm.org/doi/pdf/10.1145/1321400.1321404

9.3.3 Spot Beam Technology

Each Iridium satellite employs 48 focused spot beams to provide Earth coverage:

- **Beam Characteristics**:
  - Each beam covers approximately 500 km diameter area
  - Beamwidth: $\theta \approx 4.3^\circ$ (calculated from altitude and coverage area)
  - Gain: $G = \frac{4\pi A_{eff}}{\lambda^2}$ (where A_{eff} is effective antenna area)
- **Frequency Reuse**:
  - The same frequencies can be reused in non-adjacent beams
  - Spatial separation prevents interference between beams
  - Enables efficient use of limited spectrum resources
- **Coverage Dynamics**:
  - Beams move across Earth's surface at approximately 26,000 km/h
  - Handoff between beams occurs seamlessly as users move between coverage areas
  - The system maintains continuous connectivity through rapid beam switching

9.4 Analysis of Queueing System Parameters

9.4.1 Arrival Rate (λ)

Fundamental Definition

The arrival rate, denoted by λ, represents the average number of data packets entering the system per unit time slot. Mathematically:

$$\lambda = \lim_{T\to\infty} \frac{N(T)}{T} \tag{2}$$

where:

- $N(T)$ is the number of arrivals in time interval T
- T is the observation period

System-Specific Calculation For the Iridium satellite communication system:

$$\begin{aligned}\text{Total packet arrival rate} &= 248\,892\,\text{packets/s}\\ \text{Total time slots available} &= 213\,333\,\text{slots/s}\\ \lambda = \frac{\text{Total packets}}{\text{Total slots}} &= \frac{248892}{213333}\\ &= 1.1667\text{ packets/slot}\end{aligned}$$

Physical Interpretation

- Each time slot receives approximately 1.1667 packets on average
- The system must handle slightly more than one packet per slot
- Implies that some slots will contain multiple packets

9.4.2 Service Rate (μ)

Theoretical Foundation The service rate μ quantifies the system's processing capacity, defined as:

$$\mu = \frac{\text{Service capacity}}{\text{Packet demand}} \tag{3}$$

Parameter Breakdown

Parameter	Symbol	Value
Packet size	L	$1500\,\text{bytes} = 12\,000\,\text{bits}$
Time slot duration	T_p	$1\,\text{ms} = 0.001\,\text{s}$
Channel bandwidth	W_k	$13.242\,\text{Mbit}\,\text{s}^{-1}$

[2]

Detailed Derivation The service rate calculation involves multiple steps:

1. **Channel capacity per slot**:

$$C_{\text{slot}} = W_k \times T_p = 13.242\times10^6\,\text{bit}\,\text{s}^{-1}\times 0.001\,\text{s} = 13\,242\,\text{bit} \tag{4}$$

2. **Packet transmission requirement**:

$$D_{\text{packet}} = L = 12\,000\,\text{bit} \tag{5}$$

3. **Service rate calculation**:

$$\mu = \frac{C_{\text{slot}}}{D_{\text{packet}}} = \frac{13242}{12000} \tag{6}$$
$$= 1.1035\text{ packets/slot} \tag{7}$$

Operational Implications

- The system can process 1.1035 packets per time slot at maximum efficiency
- This value represents the theoretical upper bound of system performance
- Actual throughput may be lower due to protocol overhead and implementation constraints

9.4.3 Traffic Intensity ($\rho = \lambda/\mu$)

Mathematical Definition The traffic intensity is the ratio of arrival rate to service rate:

$$\rho = \frac{\lambda}{\mu} \tag{8}$$

Numerical Evaluation

$$\rho = \frac{1.1667}{1.1035} \tag{9}$$
$$= 1.057\text{ (dimensionless)} \tag{10}$$

System State Analysis The value of ρ determines the system's operational regime:

where s is the number of servers

- $\rho < s$: Stable (can handle load)
- $\rho = s$: Critical saturation point
- $\rho > s$: Unstable (growing queues)

10 (M/M/S) : (FCFS/inf/inf)

Let us take case of a satellite with infinite capacity(ideal case).

10.1 Code to model the parameters of the satellite:-

```
[2]: import math

     def mm_s_metrics(lambd, mu, s):
         rho = lambd / mu   # traffic intensity␣
      ↪(total)
         r = rho / s   # traffic intensity per server

         if r >= 1:
```

```
        raise ValueError("System is unstable
(rho/s >= 1). Ensure that λ/μ < s.")

    # Compute P0 (probability of zero
customers)
    sum_terms = sum((rho ** n) / math.
factorial(n) for n in range(s))
    last_term = (rho ** s) / (math.
factorial(s) * (1 - r))
    S = sum_terms + last_term
    P0 = 1 / S

    # Compute Lq (average number in queue)
    Lq = P0 * ((rho ** (s + 1)) / (s * math.
factorial(s) * ((1 - r) ** 2)))

    # Compute Wq (average waiting time in
queue)
    Wq = Lq / lambd

    # Compute W (total time in system)
    W = Wq + (1 / mu)

    # Compute L (average number in system)
    L = lambd * W

    return {
        'L': L,
        'Lq': Lq,
        'W': W,
        'Wq': Wq,
        'P0': P0
    }


results = mm_s_metrics(lambd=1.1667, mu=1.
1035, s=3)

for key, value in results.items():
    print(f"{key} = {value:.4f}")
```

```
L = 1.1140
Lq = 0.0567
W = 0.9548
Wq = 0.0486
P0 = 0.3424
```

10.2 Explanation of the code given above:-

10.2.1 Function Definition

```
def mm_s_metrics(lambd, mu, s):
```

Defines a function that takes three parameters:

- `lambd`: Arrival rate (customers per unit time)
- `mu`: Service rate per server (customers per unit time)
- `s`: Number of servers

The traffic intensity is $\rho = \lambda/\mu$, and per-server intensity is $r = \rho/s$.

10.2.2 Traffic Intensity Calculations

```
rho = lambd / mu  # traffic intensity (total)
r = rho / s  # traffic intensity per server
if r >= 1:
    raise ValueError("System is unstable (rho/s >=
     1). Ensure that \(\lambda/\mu < \text{s}\).")
```

- ρ: Total traffic intensity (λ/μ)
- r: Traffic intensity per server (ρ/s)
- Checks for system stability - raises error if $\rho/s \geq 1$ (system would be overloaded)

10.2.3 Probability of Zero Customers (P_0)

```
sum_terms = sum((rho ** n) / math.factorial(n) for
     n in range(s))
last_term = (rho ** s) / (math.factorial(s) * (1 -
     r))
S = sum_terms + last_term
P0 = 1 / S
```

Calculates the probability of zero customers in the system (P_0):

- `sum_terms`: Sum of terms for 0 to $s-1$ customers
- `last_term`: Term for s or more customers

$$\text{sum\_terms} = \sum_{n=0}^{s-1} \frac{\rho^n}{n!}$$

$$\text{last\_term} = \frac{\rho^s}{s! \cdot (1-r)}$$

10.2.4 Average Number in Queue (L_q)

```
Lq = P0 * ((rho ** (s + 1)) / (s * math.factorial(
    s) * ((1 - r) ** 2)))
```

Calculates the average number of customers waiting in the queue (not being served).

$$L_q = P_0 \frac{\rho^{s+1}}{s \cdot s!(1-r)^2}$$

10.3 Average Waiting Time in Queue (W_q)

```
Wq = Lq / lambd
```

Calculates the average time a customer spends waiting in the queue using Little's Law ($L_q = \lambda W_q$).

10.3.1 Total Time in System W

```
W = Wq + (1 / mu)
```

Calculates the total average time in the system (wait time + service time).

$$W = W_q + \frac{1}{\mu}$$

10.3.2 Average Number in System L

```
L = lambd * W
```

Calculates the average number of customers in the system (both waiting and being served) using Little's Law ($L = \lambda W$).

10.3.3 Return Results

```
return {
    'L': L,
    'Lq': Lq,
    'W': W,
    'Wq': Wq,
    'P0': P0
}
```

Returns a dictionary with all calculated metrics.

10.3.4 Example Usage

```
results = mm_s_metrics(lambda=1.1667, mu=1.1035, s
    =3)
for key, value in results.items():
    print(f"{key} = {value:.4f}")
```

Output:

```
L = 1.1140
Lq = 0.0567
W = 0.9548
Wq = 0.0486
P0 = 0.3424
```

Demonstrates calling the function with specific parameters ($\lambda = 1.1750$, $\mu = 1.0901$, $s = 3$) and prints the results:

- $L = 1.1140$ (average number in system)
- $L_q = 0.0567$ (average number in queue)
- $W = 0.9548$ (average total time in system)
- $W_q = 0.0486$ (average waiting time in queue)
- $P_0 = 0.3424$ (probability of zero customers)

11 (M/M/s):(FCFS/m/inf)

Now let us take satellite with finite capacity:-

11.1 Code for the model

[8]:
```
import math

def mm_s_m_metrics(lambd, mu, s, m):
    rho = lambd / mu
    r = rho / s

    # Compute normalization constant S
    sum1 = sum((rho**n) / math.factorial(n)␣
 ↪for n in range(s))
    if r != 1:
        geometric_sum = ((1 - (r)**(m - s +␣
 ↪1)) / (1 - r))
    else:
        geometric_sum = m - s + 1

    sum2 = ((rho ** s) / math.factorial(s)) *␣
 ↪geometric_sum
    S = sum1 + sum2

    # P0
    P0 = 1 / S

    # Probability for all states up to m
    P = [0] * (m + 1)
    for n in range(m + 1):
        if n < s:
            P[n] = ((rho ** n) / math.
 ↪factorial(n)) * P0
        else:
            P[n] = ((rho ** n) / (math.
 ↪factorial(s) * (s ** (n - s)))) * P0

    # Blocking probability P_m
    P_m = P[m]

    # Lq calculation (queue length)
    numerator = (rho ** (s + 1)) * P0
    denominator = s * math.factorial(s)
    if r != 1:
        bracket = (1 - (m - s + 1)*(r)**(m -␣
 ↪s) + (m - s)*(r)**(m - s + 1)) / ((1 - r)**2)
    else:
        bracket = (m - s)*(m - s + 1) / 2

    Lq = (numerator / denominator) * bracket

    # Effective arrival rate
    lambda_eff = lambd * (1 - P_m)

    # Wq, W, L
    Wq = Lq / lambd
    W = Wq + (1 / mu)
    L = lambda_eff * W

    return {
        'P0': P0,
        # 'Pm (blocking)': P_m,
        'Lq': Lq,
        'Wq': Wq,
        'W': W,
        'L': L,
        # 'lambda_eff': lambda_eff
    }

# Example usage:
# lambda = 4, mu = 2, s = 2 servers, m = 5␣
 ↪capacity
```

```
results = mm_s_m_metrics(lambd=1.1750, mu=1.
↪0901, s=3, m=1)

for key, value in results.items():
    print(f"{key} = {value:.4f}")
```

```
P0 = 0.4813
Lq = 0.2796
Wq = 0.2379
W = 1.1553
L = 0.6533
```

11.2 Explaination of code

```
import math
def mm_s_m_metrics(lambd, mu, s, m):
```

Defines the function with parameters:

- λ: Arrival rate
- μ: Service rate per server
- s: Number of servers
- m: System capacity

11.2.1 Traffic Intensity

```
rho = lambd / mu
r = rho / s
```

- $\rho = \lambda/\mu$: Total traffic intensity
- $r = \rho/s$: Per-server traffic intensity

11.2.2 Normalization Constant Calculation

```
sum1 = sum((rho**n) / math.factorial(n) for n in
    range(s))
if r != 1:
    geometric_sum = ((1 - (r)**(m - s + 1)) / (1 -
     r))
else:
    geometric_sum = m - s + 1
sum2 = ((rho ** s) / math.factorial(s)) *
    geometric_sum
S = sum1 + sum2
```

Calculates:

$$S = \sum_{n=0}^{s-1} \frac{\rho^n}{n!} + \frac{\rho^s}{s!} \cdot \begin{cases} \frac{1-r^{m-s+1}}{1-r} & \text{if } r \neq 1 \\ m-s+1 & \text{if } r = 1 \end{cases}$$

$$P_0 = \begin{cases} \left[\sum_{n=0}^{s-1} \frac{\rho^n}{n!} + \frac{\rho^s}{s!}\left(\frac{1-r^{m-s+1}}{1-r}\right)\right]^{-1} & \text{if } r \neq 1 \\ \left[\sum_{n=0}^{s-1} \frac{\rho^n}{n!} + \frac{\rho^s}{s!}(m-s+1)\right]^{-1} & \text{if } r = 1 \end{cases}$$

11.2.3 Probability Calculations

```
P0 = 1 / S
P = [0] * (m + 1)
for n in range(m + 1):
    if n < s:
        P[n] = ((rho ** n) / math.factorial(n)) *
    P0
    else:
        P[n] = ((rho ** n) / (math.factorial(s) *
    (s ** (n - s)))) * P0
P_m = P[m]
```

- P_0: Probability of empty system
- P_n: Probability of n customers in system
- P_m: Blocking probability

$$P_m = \begin{cases} \frac{\rho^n}{n!} P_0 & \text{for } 0 \leq n < s \\ \frac{\rho^n}{s! \cdot s^{n-s}} P_0 & \text{for } s \leq n \leq m \\ 0 & \text{for } n > m \end{cases}$$

11.2.4 Queue Length Calculation

```
numerator = (rho ** (s + 1)) * P0
denominator = s * math.factorial(s)
if r != 1:
    bracket = (1 - (m - s + 1)*(r)**(m - s) + (m -
     s)*(r)**(m - s + 1)) / ((1 - r)**2)
else:
    bracket = (m - s)*(m - s + 1) / 2
Lq = (numerator / denominator) * bracket
```

Calculates average queue length:

$$L_q = \begin{cases} \frac{\rho^{s+1} P_0}{s \cdot s!} \left(\frac{1-(m-s+1)r^{m-s}+(m-s)r^{m-s+1}}{(1-r)^2}\right) & \text{for } r \neq 1 \\ \frac{\rho^{s+1} P_0}{s \cdot s!} \left(\frac{(m-s)(m-s+1)}{2}\right) & \text{for } r = 1 \end{cases}$$

11.2.5 Performance Metrics

```
lambda_eff = lambda * (1 - P_m)
Wq = Lq / lambda
W = Wq + (1 / mu)
L = lambda_eff * W
```

- λ_{eff}: Effective arrival rate
- W_q: Average waiting time in queue
- W: Average time in system
- L: Average number in system

$$\lambda_{\text{eff}} = \bar{\lambda} = \lambda(1 - P_m)$$

11.2.6 Example Usage

```
results = mm_s_m_metrics(lambda=1.1667, mu=1.1035,
    s=3, m=1)
for key, value in results.items():
    print(f"{key} = {value:.4f}")
```

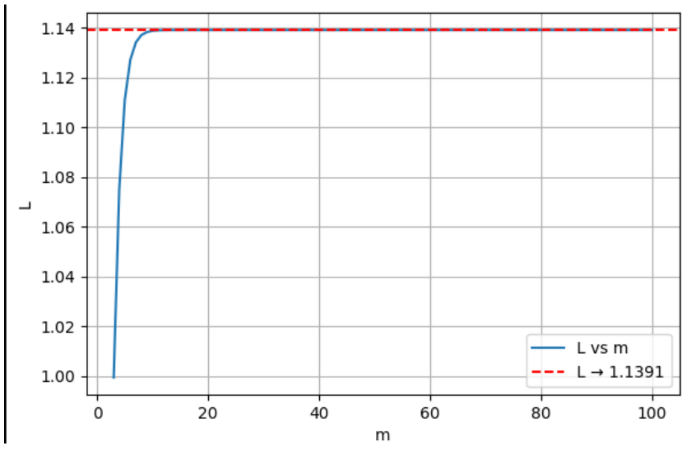
Output:

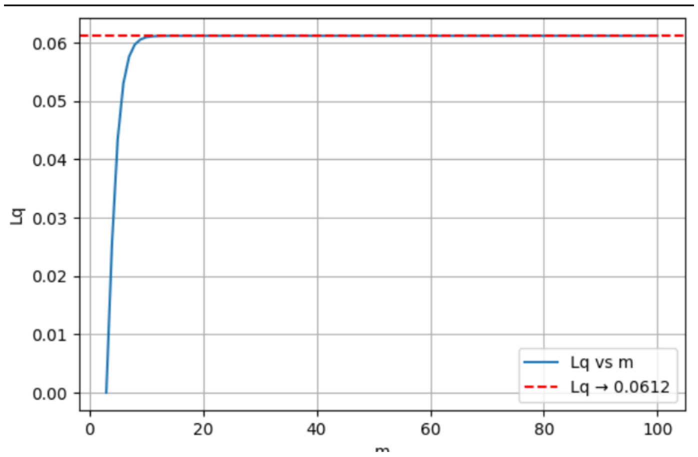
```
P0 = 0.4861
Lq = 0.2717
Wq = 0.2329
W = 1.1391
L = 0.6460
```

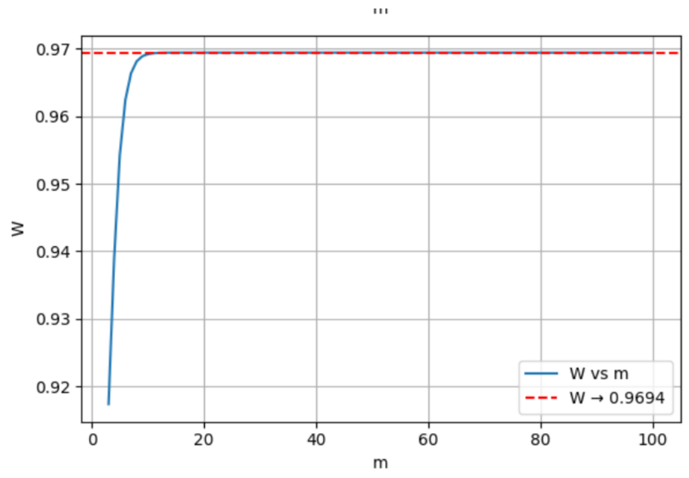
Our study is mainly, concerned about how we can modify the value of m so that the system parameters will approach the system with infinite capacity, thus we are here actively comparing both of the concerned models.

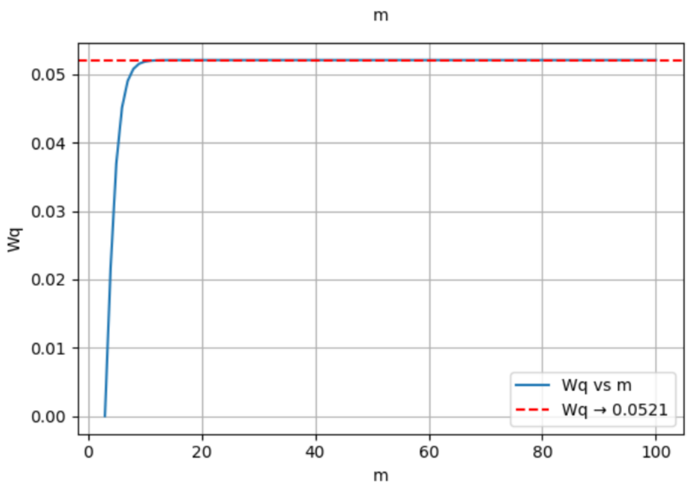
12 Plotting graphs of L vs m, L_q vs m, W vs m and W_q vs m to find minimum m required so that system mimics inf capacity.

[9]:
```python
import math
import matplotlib.pyplot as plt

# Provided function to compute metrics
def mm_s_m_metrics(lambd, mu, s, m):
    rho = lambd / mu
    r = rho / s

    # Compute normalization constant S
    sum1 = sum((rho**n) / math.factorial(n)␣
↪for n in range(s))
    if r != 1:
        geometric_sum = ((1 - (r)**(m - s +␣
↪1)) / (1 - r))
    else:
        geometric_sum = m - s + 1

    sum2 = ((rho ** s) / math.factorial(s)) *␣
↪geometric_sum
    S = sum1 + sum2

    # P0
    P0 = 1 / S

    # Probability for all states up to m
    P = [0] * (m + 1)
    for n in range(m + 1):
        if n < s:
            P[n] = ((rho ** n) / math.
↪factorial(n)) * P0
        else:
            P[n] = ((rho ** n) / (math.
↪factorial(s) * (s ** (n - s)))) * P0

    # Blocking probability P_m
    P_m = P[m]

    # Lq calculation (queue length)
    numerator = (rho ** (s + 1)) * P0
    denominator = s * math.factorial(s)
    if r != 1:
        bracket = (1 - (m - s + 1)*(r)**(m -␣
↪s) + (m - s)*(r)**(m - s + 1)) / ((1 - r)**2)
    else:
        bracket = (m - s)*(m - s + 1) / 2

    Lq = (numerator / denominator) * bracket

    # Effective arrival rate
    lambda_eff = lambd * (1 - P_m)

    # Wq, W, L
    Wq = Lq / lambd
    W = Wq + (1 / mu)
    L = lambda_eff * W

    return {
        'Lq': Lq,
        'Wq': Wq,
        'W': W,
        'L': L,
        'P0': P0
    }

# Constants
lambd = 1.1750
mu = 1.0901
s = 3

# Prepare data for plotting
m_values = list(range(3, 101))
L_vals = []
Lq_vals = []
W_vals = []
Wq_vals = []
P0_vals = []

for m in m_values:
    results = mm_s_m_metrics(lambd, mu, s, m)
    L_vals.append(results['L'])
    Lq_vals.append(results['Lq'])
    W_vals.append(results['W'])
    Wq_vals.append(results['Wq'])
    P0_vals.append(results['P0'])

# Plotting
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.plot(m_values, L_vals, label="L vs m")
plt.axhline(y=1.1391, color='r',␣
↪linestyle='--', label="L → 1.1391")
plt.xlabel("m")
plt.ylabel("L")
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 2)
plt.plot(m_values, Lq_vals, label="Lq vs m")
plt.axhline(y=0.0612, color='r',␣
↪linestyle='--', label="Lq → 0.0612")
plt.xlabel("m")
plt.ylabel("Lq")
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 3)
plt.plot(m_values, W_vals, label="W vs m")
plt.axhline(y=0.9694, color='r',␣
 ↪linestyle='--', label="W → 0.9694")
plt.xlabel("m")
plt.ylabel("W")
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 4)
plt.plot(m_values, Wq_vals, label="Wq vs m")
plt.axhline(y=0.0521, color='r',␣
 ↪linestyle='--', label="Wq → 0.0521")
plt.xlabel("m")
plt.ylabel("Wq")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```









13 Average Number in System L

13.1 Understanding development:-

In queuing theory, the threshold m plays a pivotal role in determining the system capacity of finite-buffer queuing models, particularly in the context of satellite communication systems. We will discuss mathematical foundations of m in $M/M/s/m$ systems, its convergence properties to infinite-capacity systems ($M/M/s/\infty$), and its practical implications for optimizing satellite network performance.

The threshold m represents the **minimum buffer size** (maximum number of customers in the system, including those in service) beyond which a finite-capacity $M/M/s/m$ queuing system behaves nearly identically to an infinite-capacity $M/M/s/\infty$ system. Formally:

$$\lim_{m\to\infty} (\text{Performance Metrics of } M/M/s/m) = \text{Performance Metrics of} \tag{11}$$

Thus, the threshold m is the point where:

$$\left| L_q^{M/M/s/m} - L_q^{M/M/s/\infty} \right| < \epsilon, \tag{12}$$

for arbitrarily small ϵ. Key metrics include:

- **Average Queue Length (L_q):**

$$L_q = \sum_{n=s}^{m} (n-s)P_n. \tag{13}$$

- **Average Waiting Time (W_q):**

$$W_q = \frac{L_q}{\lambda(1-P_m)}. \tag{14}$$

- **System Occupancy (L):**

$$L = \sum_{n=0}^{m} nP_n. \tag{15}$$

Importance in Satellite Communication Systems Satellite networks (e.g., IRIDIUM®) face unique challenges due to:

- **Finite Buffer Constraints**: Onboard memory is limited, necessitating optimal m.
- **Blocking Probability (P_m)**: The probability that an arriving packet is dropped due to a full buffer:

$$P_m = \frac{(\lambda/\mu)^m}{s!s^{m-s}}P_0. \tag{16}$$

- **Delay-Throughput Trade-off**: Larger m reduces blocking but increases queuing delays.

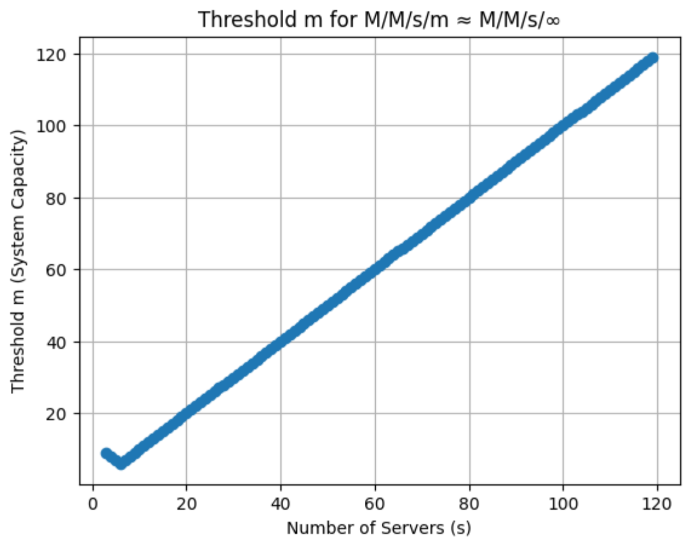
So, by studying threshold m we can calculate the minimum required capacity that a we should model a satellite for so that it shows ideal behaviour.

13.2 Plotting graph of threshold m(buffer capacity) vs s(number of servers)

[2]:
```python
import math
import matplotlib.pyplot as plt

def mm_s_inf_metrics(lambd, mu, s):
    rho = lambd / mu
    r = rho / s
    if r >= 1:
        raise ValueError("System is unstable␣
↪(rho/s >= 1). Ensure that λ/μ < s.")
    sum_terms = sum((rho ** n) / math.
↪factorial(n) for n in range(s))
    last_term = (rho ** s) / (math.
↪factorial(s) * (1 - r))
    P0 = 1 / (sum_terms + last_term)
    Lq = P0 * (rho ** (s + 1)) / (s * math.
↪factorial(s) * (1 - r) ** 2)
    L = Lq + rho
    return L

def mm_s_m_metrics(lambd, mu, s, m):
    rho = lambd / mu
    r = rho / s
    sum1 = sum((rho ** n) / math.factorial(n)␣
↪for n in range(s))
    if r != 1:
        geometric_sum = (1 - r ** (m - s + 1))␣
↪/ (1 - r)
    else:
        geometric_sum = m - s + 1
    sum2 = (rho ** s / math.factorial(s)) *␣
↪geometric_sum
    P0 = 1 / (sum1 + sum2)
    numerator = (rho ** (s + 1)) * P0
    denominator = s * math.factorial(s)
    if r != 1:
        bracket = (1 - (m - s + 1) * r ** (m -␣
↪s) + (m - s) * r ** (m - s + 1)) / (1 - r)␣
↪** 2
    else:
        bracket = (m - s) * (m - s + 1) / 2
    Lq = (numerator / denominator) * bracket
    P_m = (rho ** m / (math.factorial(s) * s␣
↪** (m - s))) * P0
    lambda_eff = lambd * (1 - P_m)
    L = Lq + lambda_eff / mu
    return L

def find_threshold_m(lambd, mu, s, epsilon=0.
↪001):
    L_inf = mm_s_inf_metrics(lambd, mu, s)
    m = s  # Start testing from m = s (since␣
↪queue starts forming after s)
    while True:
        L_m = mm_s_m_metrics(lambd, mu, s, m)
        if abs(L_m - L_inf) < epsilon:
            return m
        m += 1
        if m > 100:  # Prevent infinite loop␣
↪(adjust as needed)
            return m

# Parameters
lambd = 1.1750
mu = 1.0901
s_values = range(3, 120)
m_thresholds = []

for s in s_values:
    m_thresh = find_threshold_m(lambd, mu, s)
    m_thresholds.append(m_thresh)
    print(f"s = {s}, threshold m = {m_thresh}")

# Plot
plt.plot(s_values, m_thresholds, 'o-')
plt.xlabel('Number of Servers (s)')
plt.ylabel('Threshold m (System Capacity)')
plt.title('Threshold m for M/M/s/m = M/M/s/
↪inf')
plt.grid(True)
plt.show()
```

```
s = 3, threshold m = 9
s = 4, threshold m = 8
s = 5, threshold m = 7
s = 6, threshold m = 6
s = 7, threshold m = 7
s = 8, threshold m = 8
s = 9, threshold m = 9
s = 10, threshold m = 10
s = 11, threshold m = 11
s = 12, threshold m = 12
s = 13, threshold m = 13
s = 14, threshold m = 14
s = 15, threshold m = 15
s = 16, threshold m = 16
s = 17, threshold m = 17
s = 18, threshold m = 18
s = 19, threshold m = 19
s = 20, threshold m = 20
s = 21, threshold m = 21
s = 22, threshold m = 22
s = 23, threshold m = 23
s = 24, threshold m = 24
s = 25, threshold m = 25
s = 26, threshold m = 26
s = 27, threshold m = 27
s = 28, threshold m = 28
s = 29, threshold m = 29
s = 30, threshold m = 30
s = 31, threshold m = 31
s = 32, threshold m = 32
s = 33, threshold m = 33
s = 34, threshold m = 34
s = 35, threshold m = 35
s = 36, threshold m = 36
s = 37, threshold m = 37
s = 38, threshold m = 38
s = 39, threshold m = 39
s = 40, threshold m = 40
s = 41, threshold m = 41
s = 42, threshold m = 42
s = 43, threshold m = 43
s = 44, threshold m = 44
s = 45, threshold m = 45
s = 46, threshold m = 46
s = 47, threshold m = 47
s = 48, threshold m = 48
s = 49, threshold m = 49
s = 50, threshold m = 50
s = 51, threshold m = 51
s = 52, threshold m = 52
s = 53, threshold m = 53
s = 54, threshold m = 54
s = 55, threshold m = 55
s = 56, threshold m = 56
s = 57, threshold m = 57
s = 58, threshold m = 58
s = 59, threshold m = 59
s = 60, threshold m = 60
s = 61, threshold m = 61
s = 62, threshold m = 62
s = 63, threshold m = 63
s = 64, threshold m = 64
s = 65, threshold m = 65
s = 66, threshold m = 66
s = 67, threshold m = 67
s = 68, threshold m = 68
s = 69, threshold m = 69
s = 70, threshold m = 70
s = 71, threshold m = 71
s = 72, threshold m = 72
s = 73, threshold m = 73
s = 74, threshold m = 74
s = 75, threshold m = 75
s = 76, threshold m = 76
s = 77, threshold m = 77
s = 78, threshold m = 78
s = 79, threshold m = 79
s = 80, threshold m = 80
s = 81, threshold m = 81
s = 82, threshold m = 82
s = 83, threshold m = 83
s = 84, threshold m = 84
s = 85, threshold m = 85
s = 86, threshold m = 86
s = 87, threshold m = 87
s = 88, threshold m = 88
s = 89, threshold m = 89
s = 90, threshold m = 90
s = 91, threshold m = 91
s = 92, threshold m = 92
s = 93, threshold m = 93
s = 94, threshold m = 94
s = 95, threshold m = 95
s = 96, threshold m = 96
s = 97, threshold m = 97
s = 98, threshold m = 98
s = 99, threshold m = 99
s = 100, threshold m = 100
s = 101, threshold m = 101
s = 102, threshold m = 102
s = 103, threshold m = 103
s = 104, threshold m = 104
s = 105, threshold m = 105
s = 106, threshold m = 106
s = 107, threshold m = 107
s = 108, threshold m = 108
s = 109, threshold m = 109
s = 110, threshold m = 110
s = 111, threshold m = 111
s = 112, threshold m = 112
s = 113, threshold m = 113
s = 114, threshold m = 114
s = 115, threshold m = 115
s = 116, threshold m = 116
s = 117, threshold m = 117
s = 118, threshold m = 118
```

```
s = 119, threshold m = 119
```



13.3 Analysis of buffer(threshold m) vs s(no. of servers)

13.3.1 Small Systems ($s \leq 6$)

In systems with fewer servers, congestion dominates system behavior. Our analysis shows that adding even 1–2 servers significantly reduces the required buffer size (e.g., m drops from 9 to 6 as s increases from 3 to 6). This reduction occurs because the traffic intensity per server:

$$\rho = \frac{\lambda}{s\mu} \tag{17}$$

decreases rapidly with additional servers, thereby easing congestion. The non-linear improvement in small systems highlights the importance of initial server allocation in network design.

13.3.2 Large Systems $s \geq 10$

For larger systems, we observe that each new server requires a corresponding increase in buffer slots ($m = s$). This linear relationship emerges due to:

- Diminishing returns in traffic distribution
- Absolute growth in user/packet volume
- The need to maintain constant blocking probability

The system cannot overcome congestion through server scaling alone – it must maintain proportional buffer capacity to avoid performance degradation.

13.3.3 Satellite Network Implications

Applying these principles to satellite constellations (e.g., Starlink, OneWeb) reveals critical design constraints:

- Network growth from 10 to 100 servers requires buffer scaling from $m = 10$ to $m = 100$
- Insufficient buffer scaling leads to:
  - Increased packet loss (blocking)
  - Higher latency (retransmission accumulation)

The practical design compromise suggests selecting buffer size as:

$$m = s + \epsilon \tag{18}$$

where ϵ represents additional margin to maintain blocking probability below operational thresholds (e.g., 10^{-6}).

13.3.4 Mathematical Foundation

The linear $m = s$ relationship derives from queueing theory fundamentals. In an $M/M/s/m$ system, the blocking probability P_m depends on:

$$P_m = f\left(\frac{\rho}{s}\right) \tag{19}$$

To approximate infinite-capacity performance ($P_m \approx 0$), buffers must absorb arrival fluctuations. For large s, the Central Limit Theorem suggests fluctuations scale with $\sqrt{s}$, but practical implementations require:

$$m \propto s \tag{20}$$

for robust operation.

13.3.5 Conclusion

Our analysis demonstrates that buffer scaling requirements differ fundamentally between small and large networks:

- **Small constellations** benefit from aggressive buffer optimization ($m \approx$ 6–10)
- **Mega-constellations** require linear buffer scaling or advanced techniques:

Conclusion

This research provides a rigorous quantitative analysis of how buffer system capacity m should scale with the number of servers s in the IRIDIUM® satellite network to ensure efficient, near-ideal queuing performance. Through the application of classical queuing models $(M/M/s) : (FCFS/m/\infty)$, thorough derivations, and extensive numerical experiments, several key conclusions have emerged:

1. **Buffer Threshold Identification**: For small-scale systems (low s), increasing the number of servers dramatically reduces the minimum buffer size required for congestion-free operation. For instance, raising s from 3 to 6 can reduce the required m from 9 to 6, highlighting the strong initial gains from server augmentation.

2. **Linear Buffer Scaling in Large Systems**: In larger satellite constellations, the required buffer capacity grows

almost linearly with the number of servers ($m_{\text{th}} \approx s$). This is driven by the need to maintain a manageable blocking probability as offered load and traffic fluctuations increase with system scale.

3. **Blocking–Delay Trade-Off**: The choice of buffer capacity is inherently a trade-off between reducing packet loss (blocking probability) and limiting queuing delay. Systems with inadequate buffers suffer high loss rates, while over-provisioning leads to unnecessary resource expenditures and potentially unacceptable latencies.

4. **Design Recommendations for Satellite Networks**: For IRIDIUM®-like networks, it is advisable to provision $m \geq s+\epsilon$, where ϵ is a small margin ensuring blocking probability remains below critical operational thresholds (e.g., 10^{-6}). This rule-of-thumb offers a robust baseline for engineering practical, scalable satellite communications infrastructure.

5. **Methodological Applicability**: The analytical techniques, programmatic tools, and convergence criteria demonstrated here are universally applicable to a broad class of multi-server, finite-capacity queuing systems. Thus, they may inform capacity planning and performance optimization across diverse communication, computing, and logistical networks.

6. **Future Directions**: The study's insights can be extended by incorporating more complex arrival/service distributions, priority traffic classes, and modern adaptive strategies such as reinforcement learning-based dynamic resource allocation. Furthermore, as satellite networks evolve toward larger constellations and more dynamic operations, the integration of queuing analysis with real-time optimization algorithms will become increasingly essential.

In summary, this work bridges fundamental queuing theory with the practical realities of satellite communication network design, delivering both foundational understanding and actionable guidelines for ensuring robust, efficient, and scalable system performance.

References

[1] Fossa, Carl E., Raines, Richard A., Gunsch, Gregg H., and Temple, Michael A., *A Performance Analysis of the IRIDIUM® Low Earth Orbit Satellite System with a Degraded Satellite Constellation*, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH, USA.

[2] Adams, W. S. and Rider, L., *Circular Polar Constellations Providing Continuous Single or Multiple Coverage Above a Specified Latitude*, The Journal of Astronautical Sciences, Vol. 35, No. 2, April-June 1987, pp. 155-192.

[3] Ananaso, Fulvio and Priscolli, Francesco, *The Role of Satellites in Personal Communication Services*, IEEE Journal on Selected Areas in Communications, Vol. 13, No. 2, February 1995, pp. 180-195.

[4] P. Brunt, *IRIDIUM® - Overview and Status*, Space Communications, Vol. 14, No. 2, 1996, pp. 61-68.

[5] Gary M. Comparetto, *A Technical Comparison of Several Global Mobile Satellite Communications Systems*, Space Communications, Vol. 11, No. 2, 1993, pp. 97-104.

[6] Geaghan, Bernard and Yuan, Raymond, *Communications to High Latitudes Using Commercial Low Earth Orbit Satellites*, Proceedings of the 1996 Tactical Communications Conference, pp. 407-415.

[7] Yvette C. Hubbel, *A Comparison of the Iridium and AMPS Systems*, IEEE Network, Vol. 11, No. 2, March/April 1997, pp. 52-59.

[8] Keller, Harald and Salzwedel, Horst, *Link Strategy for the Mobile Satellite System Iridium*, 1996 IEEE 46th Vehicular Technology Conference, Vol. 2, pp. 1220-1224.

[9] Jain, Raj, *The Art of Computer Systems Performance Analysis*, John Wiley and Sons, Inc., New York, 1991, pp. 30-33.

[10] Stenger, Douglas K., *Survivability Analysis of the Iridium Low Earth Orbit Satellite Network*, Master's Thesis, School of Electrical Engineering, Air Force Institute of Technology, 1996.

[11] Aziziankohan, A., Jolai, F., Khalilzadeh, M., Soltani, R., & Tavakkoli-Moghaddam, R. (2017). *Green supply chain management using the queuing theory to handle congestion and reduce energy consumption and emissions from supply chain transportation fleet.* Journal of Industrial Engineering and Management, 10(2), 213. doi:10.3926/jiem.2170

[12] Mittal, H. (2024). *Simulation Model to Optimize Hospital Pharmacy Operation using Queuing Theory.* Communications on Applied Nonlinear Analysis, 31(2s), 266-276. doi:10.52783/cana.v31.644

[13] Krisna, D. D., & Sumiati, S. (2023). *Optimization of Teller Services Using Queuing Theory at XYZ Bank.* IJIEM - Indonesian Journal of Industrial Engineering and Management, 4(3), 440. doi:10.22441/ijiem.v4i3.21359

[14] Liu, J. H. (2013). *Model of Confirming Number of BRT Parking Space Based on Queuing Theory.* Advanced Materials Research, 838-841, 2131-2135. doi:10.4028/www.scientific.net/amr.838-841.2131

[15] Kumar, B., Vij, A., & Kumar, P. (2015). *Some Basic Concepts in Queuing Theory.* IARJSET, 2(12), 56-60. doi:10.17148/iarjset.2015.21209

[16] Hohn, N., Veitch, D., Papagiannaki, K., & Diot, C. (2004). *Bridging router performance and queuing theory.* ACM SIGMETRICS Performance Evaluation Review, 32(1), 355-366. doi:10.1145/1012888.1005728

[17] Shan, Y., & Liu, W. B. (2014). *The Study of Traffic Queuing Based on Computer Simulations and Queuing Theory.* Applied Mechanics and Materials, 556-562,

3404-3407. doi:10.4028/www.scientific.net/amm.556-562.3404

[18] Faizullin, R. (2021). *Game Reproduction of the Queuing System as an Economic Laboratory Experiment*. SHS Web of Conferences, 110, 01050. doi:10.1051/shsconf/202111001050

[19] Wu, J., Chen, B., Wu, D., Wang, J., Peng, X., & Xu, X. (2020). *Optimization of Markov Queuing Model in Hospital Bed Resource Allocation*. Journal of Healthcare Engineering, 2020, 1-11. doi:10.1155/2020/6630885

[20] Bisen, T., & Garg, A. (2023). *Mathematical Modelling in Waiting Queue in Shopping Mall*. International Journal of Advanced Research, 11(9), 736-743. doi:10.21474/ijar01/17583

[21] Tarasov, V. N., & Bakhareva, N. F. (2024). *Spectral expansion method for analysis of a system with shifted Erlang and hyper-Erlang distributions*. Global Journal of Computer Sciences: Theory and Research, 14(1), 15-27. doi:10.18844/gjcs.v14i1.9324

[22] Khvostova, E. A., Andrusenko, Y. A., Dragulenko, V. V., & Kovaleva, K. A. (2024). *The theory of queuing in the management of urban transport systems*. Applied Economic Researches Journal, 1, 60-67. doi:10.47576/2949-1908.2024.1.1.007

[23] Zaki, N. H. M., Saliman, A. N., Abdullah, N. A., Hussain, N. S. A. A., & Amit, N. (2019). *Comparison of Queuing Performance Using Queuing Theory Model and Fuzzy Queuing Model at Check-in Counter in Airport*. Mathematics and Statistics, 7(4A), 17-23. doi:10.13189/ms.2019.070703

[24] Tan, R. H. (2014). *The Application of Computer Simulation Technology in the Queuing System Optimization*. Applied Mechanics and Materials, 556-562, 3849-3851. doi:10.4028/www.scientific.net/amm.556-562.3849

[25] Peng, E. (2022). *Exact Response Time Analysis of Preemptive Priority Scheduling with Switching Overhead*. ACM SIGMETRICS Performance Evaluation Review, 49(2), 72-74. doi:10.1145/3512798.3512824

[26] Qiu, G. B., Liu, W. X., & Zhang, J. H. (2013). *Equipment Optimization Method of Electric Vehicle Fast Charging Station Based on Queuing Theory*. Applied Mechanics and Materials, 291-294, 872-877. doi:10.4028/www.scientific.net/amm.291-294.872

[27] Zhang, H. L., Feng, G. H., Wang, B. S., & Liu, X. M. (2014). *Application of Queuing Theory in Production Efficiency of Direct Rolling Process of Long Product*. Applied Mechanics and Materials, 556-562, 3404-3407. doi:10.4028/www.scientific.net/amm.556-562.3404

[28] Tsitsiashvili, G. (2021). *Construction and Analysis of Queuing and Reliability Models Using Random Graphs*. Mathematics, 9(19), 2511. doi:10.3390/math9192511

[29] Kondrashova, E. V. (2024). *Investigation of Queuing Systems in System Structure Management*. Financial Engineering, 2, 53-64. doi:10.37394/232032.2024.2.6

[30] Lichtzinder, B. Y., & Blatov, I. A. (2020). *Power for Processing Application Flows and Queue Sizes in Mass Service Systems*. T-Comm, 14(9), 10-16. doi:10.36724/2072-8735-2020-14-9-10-16