MONASH University
Information Technology

FIT3142 Distributed Computing

Topic 9: Distributed Application Performance Modelling Part 2

Dr Carlo Kopp
Faculty of Information Technology
Monash University

Why Study Distributed Application Performance?

- Unlike applications on standalone systems, Distributed Systems can be extremely sensitive to performance problems, to the extent of making a system unusable;
- Foundation knowledge: Because Distributed Systems are now widely used, understanding their performance is essential to both design and implementation of code;
- Practical skills: When coding distributed applications you will have to design the application with due consideration to performance constraints in the algorithms used;
- Practical skills: If you do not understand performance problems in distributed systems, you might end up producing code that appears to be correct, functions as intended, but is unusable due to performance problems that cannot be fixed as they result from the behaviour of the algorithm used in a distributed system;

Amdahl's Law in Distributed Computing

- Consider Amdahl's Law:

$$S = \frac{1}{s + \frac{p}{N}}$$

- The variable *p* reflects the fraction of time processing data with no dependencies, which can be parallelised. *p* is "parallel" time.
- The variable *s* reflects the fraction of time processing data with dependencies, which cannot be parallelised. *s* is "serial" time.

Amdahl's Law in Distributed Computing

- **In a distributed environment, the fraction of time consumed performing "serial" computation comprises the time consumed by the CPUs doing the work, and the time consumed communicating, when hosts working on data with dependencies must transmit the results of communications.**
- **Therefore:**

$$s = s_{comp} + s_{network}$$

- **Where $s_{network}$ is the time for these results to be encapsulated in a message, sent across the network, and extracted for use at the destination.**

Amdahl's Law in Distributed Computing

- This results in *Amdahl's Law for Distributed Systems*:

$$S = \frac{1}{s_{comp} + s_{network} + \frac{p}{N}}$$

- What is important is that the network can impact speedup performance no differently than the fraction of time expended on computations with dependencies.
- If the network provides extremely high capacity and thus short delays to transmit, then $s_{comp} >> s_{network}$ and the distributed system behaves no differently from a conventional "bussed" multiprocessor.

Revision: The Impact of Network Performance

- The time it takes a message to travel across a network depends on a number of factors:

1) Processing delays on the middleware libraries and protocols stack at the transmitting end;
2) Cumulative queueing delays in routers along the transmission route through the network;
3) Propagation delay through cables in the network, proportional to total cable length;
4) Processing delays on the middleware libraries and protocols stack at the receiving end.

- Only 3) is constant in time, the remainder being time variant, and depend on network congestion.

Revision: The Impact of Network Performance

- The result of Amdahl's Law and network delays is that the performance of a distributed application improves with decreasing frequency of messages required, decreasing size of messages required, and decreasing network load.
- If an application is to be run distributed, it is therefore important to understand what data dependencies exist in the algorithm.
- For many problems, there will be choices in the "granularity" of the computation, i.e. the size of the "chunks" distributed across the grid.
- Increasing *N* also increases $s_{network}$, costing speed.

Modelling Grid Performance (1)

- Modelling the performance of an application intended to run on a conventional multi-processor architecture is often difficult.
- The starting point is model such as Amdahl's Law and its parametric requirement for decomposition of the computing load into "parallel" components without data dependencies, and "serial" components with dependencies.
- This typically requires careful study of the algorithm used in the computation, as well as the implementation of the algorithm, to identify what data dependencies might exist.

Modelling Grid Performance (2)

- Some application environments or compilers can parallelise automatically, however this function is usually built around a particular multiprocessor architecture (more later).
- More than often this facility will not be available and it will be necessary to analyse the code and break it into modules or functions which are suitable for parallel execution, and vice versa.
- Code with data dependencies behaves as the serial component in the Amdahl model.

Modelling Grid Performance (3)

- In a clustered or grid environment, the serial component comprises both computation time, and time to transmit results across the local network, a dedicated high speed network, or the Internet.
- The computational component of that time can be easily benchmarked for the processor and compiler intended to be used. Alternately, it could be treated analytically.
- For a given processor architecture and compiler, clock speed, etc, the time for such computations is often (nearly) constant.
- Approximating s_{comp} ~ const is often acceptable.

Modelling Grid Performance - Network

- In clusters or grids with dedicated high performance networks, where $s_{comp} >> s_{network}$, the time consumed in transmission of results between processors can safely be approximated by using a constant value, typically measured by benchmarking.
- This approach breaks down under one or both of these conditions:

1 $s_{comp} \approx s_{network}$

2 $s_{comp} << s_{network}$

Modelling Grid Performance - Network

- **The problem is usually complicated by the variability of transmission time over the network, i.e. $s_{network}$ *(t)*.**
- ***This variability depends on the instantaneous congestion of the network and the size of the data being transmitted.***
- **Instantaneous network congestion can be modelled using queueing theory, which is also one of the most common techniques used for modelling performance in large multiuser machines.**
- **Queueing models are therefore important to know.**

Modelling Grid Performance - Network

- **Variability arising from variable data volumes can be readily measured or modelled, given knowledge of the program's behaviour.**
- **Importantly, network performance will impact behaviour, and again queueing theory applies.**
- **What Amdahl's Law shows is that grid application performance is degraded the most where either:**
  - There is a lot of data dependency in the computation.
  - There are a large number of CPUs required for the calculation.
  - The network is slow or highly variable in throughput performance.

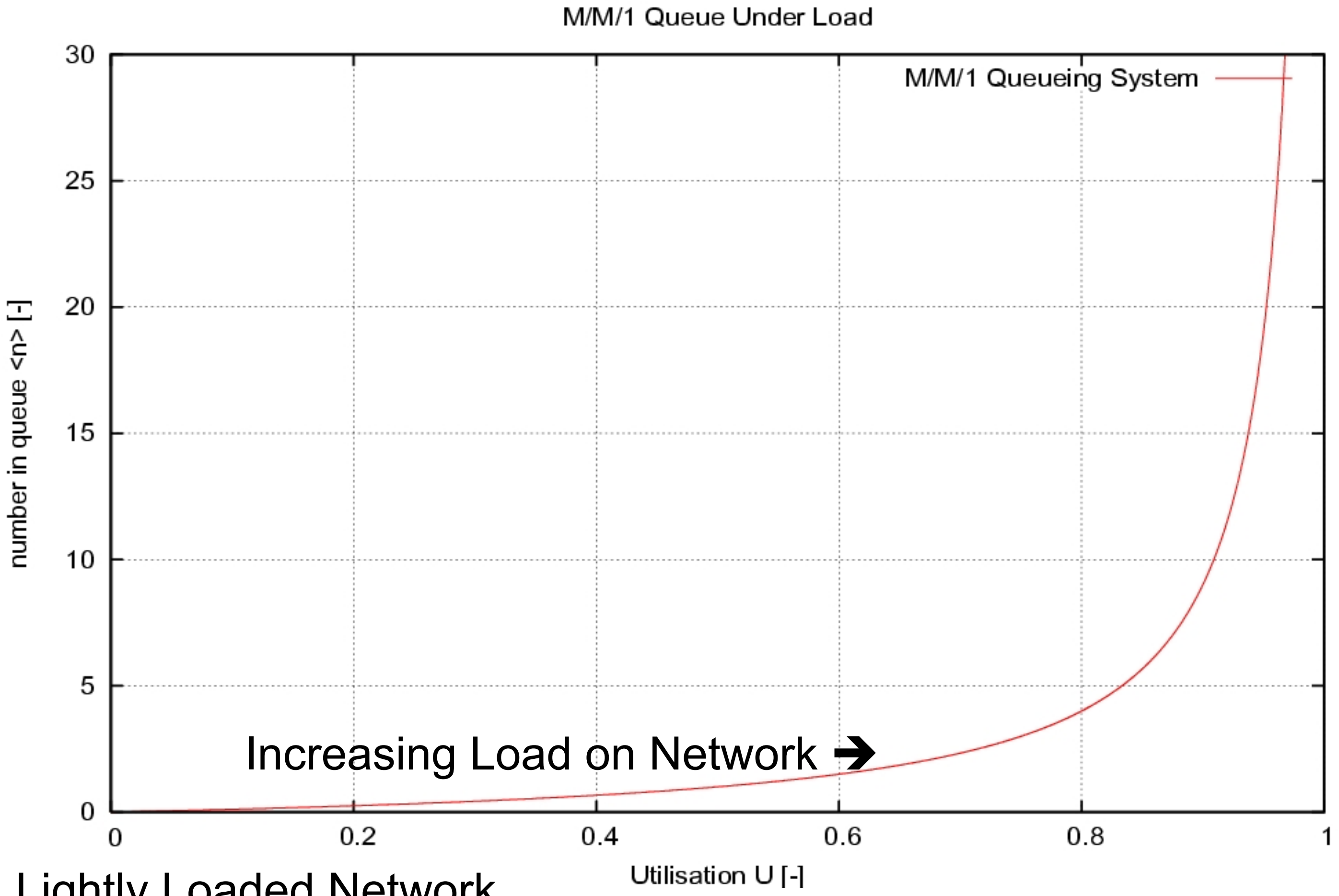
Queueing Models for Networks

- A detailed treatment of queueing theory exceeds the scope of this subject [Non-examinable tutorial notes have been provided to aid in understanding how queues behave].
- In terms of understanding grid performance degradations, we are most interested in what happens in a queueing system when the load (i.e. amount of traffic) approaches the limits of the queue's capacity to handle.
- This is termed "saturation" and results in a non-linear and rapid increase in delays experienced waiting in the queue.

The Network Saturation Problem

Saturated Network

Increasing Delay Through Network ➔



Increasing Load on Network ➔

Lightly Loaded Network

Queueing Models for Networks

- **If the network is poorly matched to the grid application, third party background traffic may drive it into saturation, causing unpredictable traffic delays and thus *unpredictable* degradation in grid application performance.**
- **If the grid application produces large surges in traffic load across the network, it may drive the network into saturation, resulting in unwanted but *predictable* degradation in grid application performance.**
- **How can a grid application designer avoid these problems?**

Matching Applications and Networks

- The application designer must first understand what traffic the application produces, and to what extent transmission delays will impair application performance.
- If the application is highly sensitive to delays, then measures will be needed to control the delay through the network.
- Two basic strategies are possible:

1 Install a network with very much higher throughput.

2 Use bandwidth reservation techniques to ensure that the grid application does not have to compete for network bandwidth / capacity.

Higher Network Capacity Approach

- Guaranteed to improve performance every time.
- Simple to understand and simple to explain.
- Does not require particular manipulation of the grid application software or toolset.
- Preferred strategy where possible. *What is wrong with this strategy?*
- Increasingly expensive to implement if network spans buildings, campuses, sites, cities, states or nations.
- *The cost is driven by the cost of installing or leasing high speed optical fibre cables; the cost is directly proportional to the length and number of links.*

Bandwidth Reservation Approach (1)

- Most modern Internet Protocol networks support DiffServ (RFC 2475 etc) or IntServ (RFC1633 etc) Quality of Service (QoS) mechanisms.
- QoS mechanisms allow an application to negotiate with the network to reserve bandwidth / capacity for a connection.
- Because bandwidth / capacity is reserved, other traffic cannot interfere and introduce unwanted delays to traffic.
- Bandwidth Reservation is a cheap approach to solving network delay problems, as it involves addition or use of middleware functions.

Bandwidth Reservation Approach (2)

- **There are limitations in Bandwidth Reservation:**
  - The grid middleware in use may not support this functionality well, or at all.
  - The operating system network protocol stack may not support this functionality well, or at all.
  - The network may not support QoS functions, or do so poorly.
  - Even if Bandwidth Reservation is available, the network may not have the performance to cope.
- ***The best approach for dealing with network throughput and delay problems is always to understand and optimise the application first, then deal with the network if there is no other choice.***

Bandwidth Reservation in Grids

- Immature technology at this time, with numerous proposals or designs competing.
- General-purpose Architecture for Reservation and Allocation (GARA) by Foster, Kesselman, Lee et al. has been popular but lacks some critical functions such as Service Level Agreements (SLA) used in established QoS schemes, and is not OGSA complaint.
- GGF Grid Resource Agreement and Allocation Protocol (GRAAP) Working Group sponsoring proposals for operational grid QoS support.
- QoS still poorly supported in operational networks.

Generalising Grid Performance Models

- Amdahl's Law is one of a number of models used for predicting the performance of parallel architectures.
- Amdahl's Law is useful for explaining the behaviour of a parallel system and the impact of data dependencies and communications delays, but it is often difficult to use since determining the serial component is not always easy.
- Numerous alternative models to Amdahl exist and have been used to analyse performance of parallel applications.
- Examples include Gustafsson, Erlang, harmonic speedup, LogP and MRM.

MRM Model

- The most general model available is the MRM (Machine Repairman Model), a mathematical queueing model developed initially for modelling production lines, and used in Operations Research.

$$X(P) = \frac{P}{R(P) + Z}$$

- X- mean throughput; P – number of processors; R – interconnect latency including waiting times; Z – mean execution time in processor.
- Amdahl's Law can be directly remapped into the MRM model (Gunther - http://xxx.lanl.gov/pdf/cs/0210017v1)

Modelling Parallel Application Behaviour

- **The starting point in designing any grid application is to understand the behaviour of the algorithm and its data dependencies, as these determine what the networking needs of the application might be.**
- **The most difficult applications are those with significant data dependencies, examples being two or three dimensional mesh computations.**
- **In a mesh computation, dependencies exist between individual cells in the mesh.**
- **The result is that the partitioning of the mesh into a grid will impact efficiency and parallel speedup.**
- ***A smaller number of processors each doing more work might perform better than a larger number of processors each doing less work individually.***

Parallelisation/Partitioning of Problems

- In general, there does not appear to be any single algorithm which can produce an optimal partitioning for all possible problems.
- This is because of the diversity in the different problems to be solved, which result in different patterns of communication between processors.
- Many existing classes of applications have good partitioning algorithms available as much research has been done; others do not.
- *The first step in designing a grid application, or migrating an application to a grid, must be the analysis of how to partition the problem.*

Parallelisation/Partitioning of Problems

- **Step 1: Is there an existing partitioning algorithm or software tool for the problem/application to be run on a grid? If yes, use this method, else:**
- **Step 2: What class/category/type does the problem / application belong to, and are there similar problems for which good partitioning algorithms exist? If yes, locate the method/software and use it, else:**
- **Step 3: Analyse the behaviour of problem / application to be run on a grid, and find the best partitioning strategy, if it exists.**
- ***It is possible that good strategies do not exist.***

Reading/ References

- Neil J. Gunther, *Performance Ponderings*, URL: http://www.perfdynamics.com/papers.html
- Neil J. Gunther, *Unification Of Amdahl's Law, LogP And Other Performance Models For Message-passing architectures*, URL: http://www.perfdynamics.com/Papers/pdcs05.pdf
- Rashid J., *Analysis and Provision of QoS for Distributed Grid Applications* URL: http://www.wesc.ac.uk/resources/publications/pdf/final-jogc.pdf

Not Examinable – Recommended Reading

QUEUEING THEORY TUTORIAL

Overview

- What is a queue and why is it important?
- Queueing systems – air traffic, automobile traffic, shopping queues, production lines, networks
- Concepts in queueing theory:

1. Random processes and interarrival time
2. M/M/1 queue attributes
3. The queue saturation problem – open/closed
4. Modelling queue behaviour
5. Queued system failures – congestion collapse
6. Well designed vs poorly designed queueing systems

What is a queue?

- **Any system which exhibits queueing behaviour can be mathematically modelled as a queue.**
- **The simplest queue is a system where some entities or activities must wait to be serviced.**
- **The behaviour of queueing systems can be readily modelled with mathematics, accepting that some types of queues can be intractable, and others easy.**
- Leonard Kleinrock: **"We study the phenomena of standing, waiting, and serving, and we call this study Queueing Theory." "Any system in which arrivals place demands upon a finite capacity resource may be termed a queueing system."**

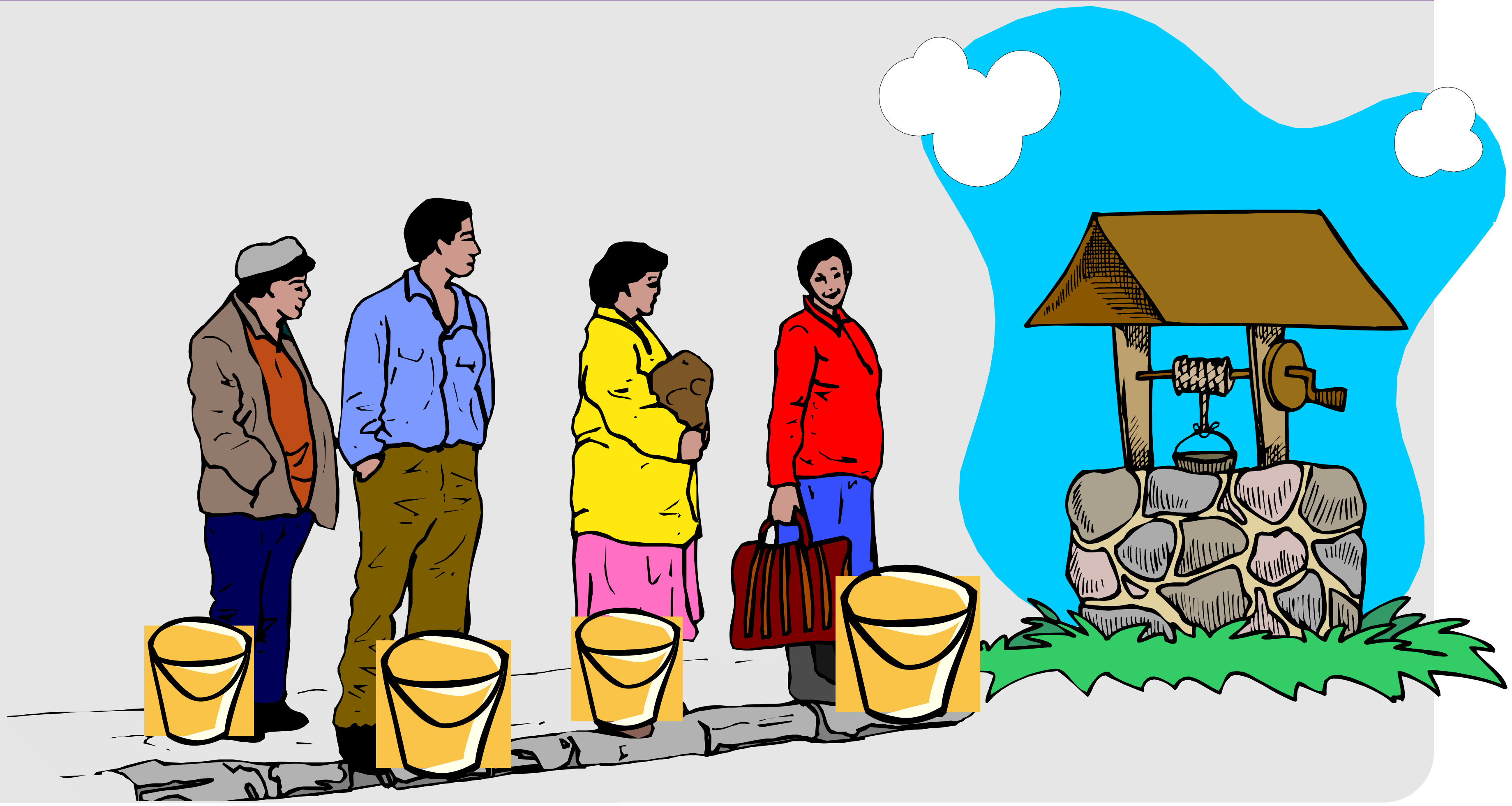
Examples of Queueing Systems

- Automobile traffic – every set of traffic lights forms a queue, in which the automobiles queue up for access to the intersection.
- Production lines – production workpieces flow through a line and must queue for every production task to be done.
- Shopping queues – shops, banks, etc. Customers queue up for access to tellers or checkout operators.
- Telephone switches – subscribers must queue for access to lines out of the exchange.
- Packet switch networks – packets queue at routers.

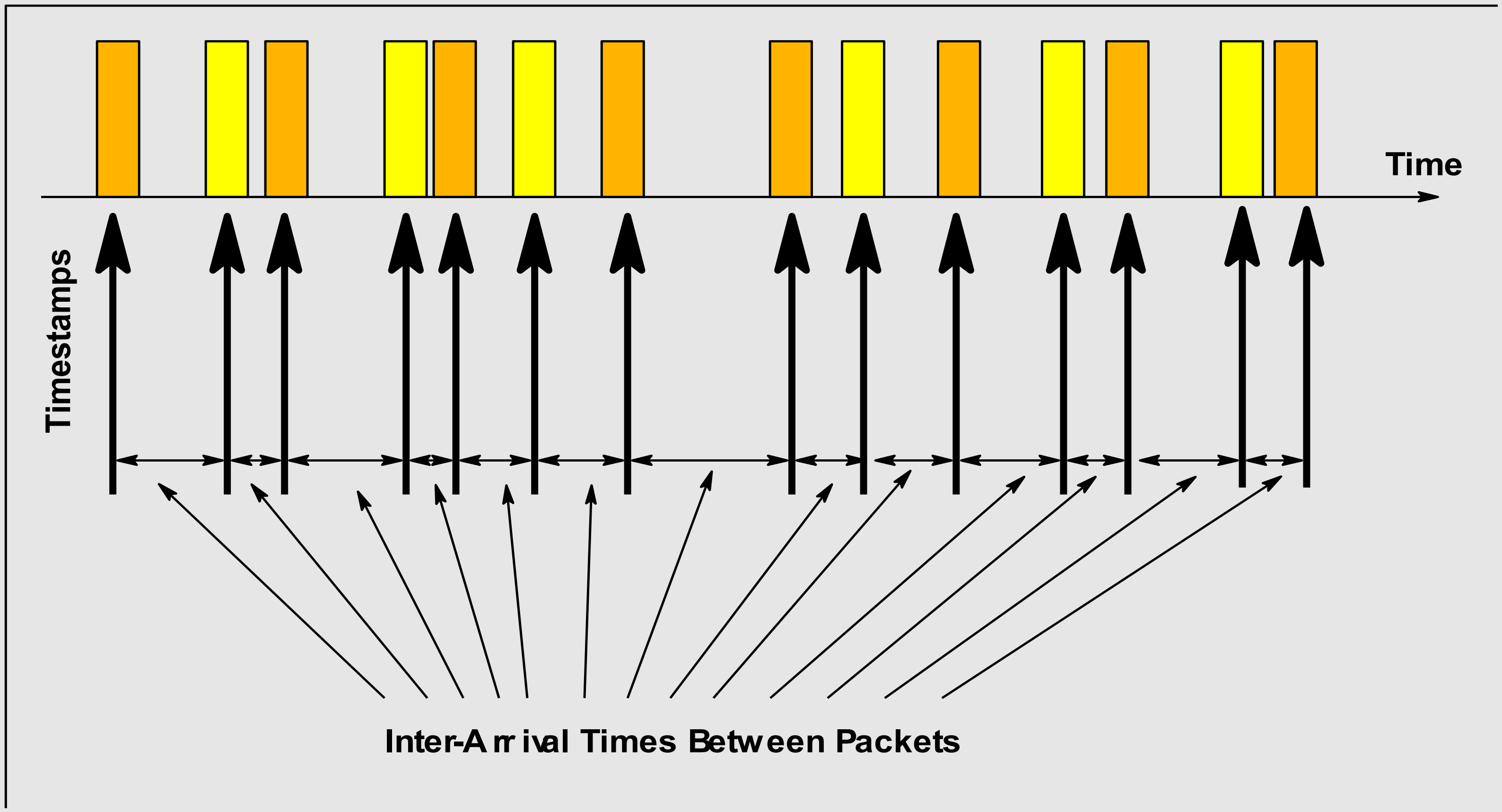
A Trivial Queueing System

- Imagine a village with a single well, which uses a pulley system to lower and raise a bucket.
- Villagers must queue at the well to fill their buckets, barrels or bags with water.
- If every villager takes the same amount of time to fill a container, how many villagers are in the queue?
- How long must the villagers stand in the queue?
- What happens if the villagers arrive at random times?
- What happens if the villagers have random sized containers to fill?
- How long is the queue, and how much time must the villagers spend waiting in the queue?

A Trivial Queueing System



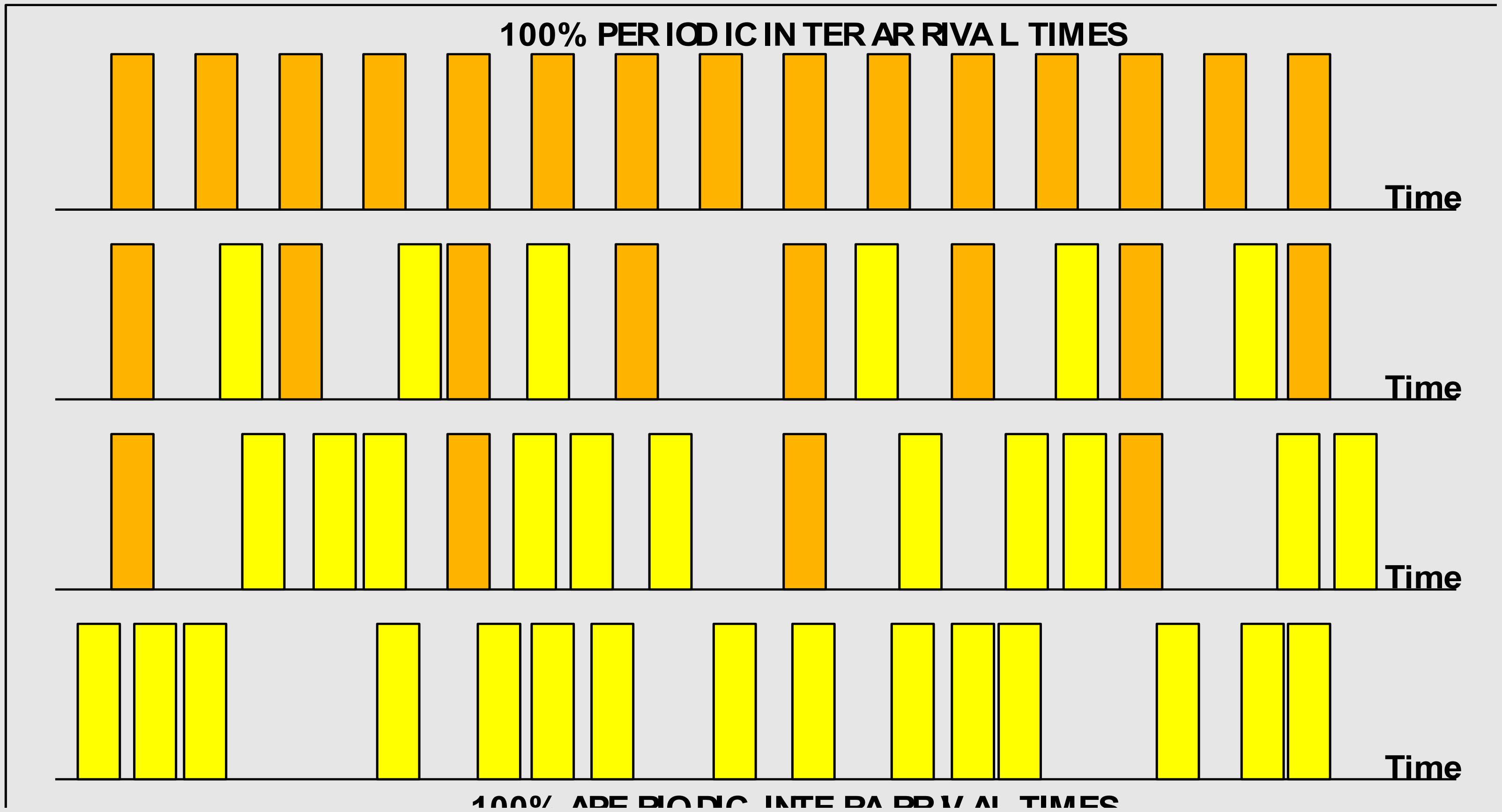
Interarrival Times



Queuing Theory Concepts (1)

- All packet switching and routing networks by their nature employ buffering mechanisms to store incoming packets, which are then forwarded to other switches or routers.
- Any ’store and forward’ buffering system behaves as a queue, and is thus subject to the mathematical properties of queueing systems.
- To properly understand the behaviour of circuit and packet switching systems, we must have a good understanding of how queues behave under increasing load.
- In all such systems incoming packets arriving at the switch or router will typically arrive at unequal time intervals, or ’inter-arrival times’. The same is true of telephone calls arriving at a switch.

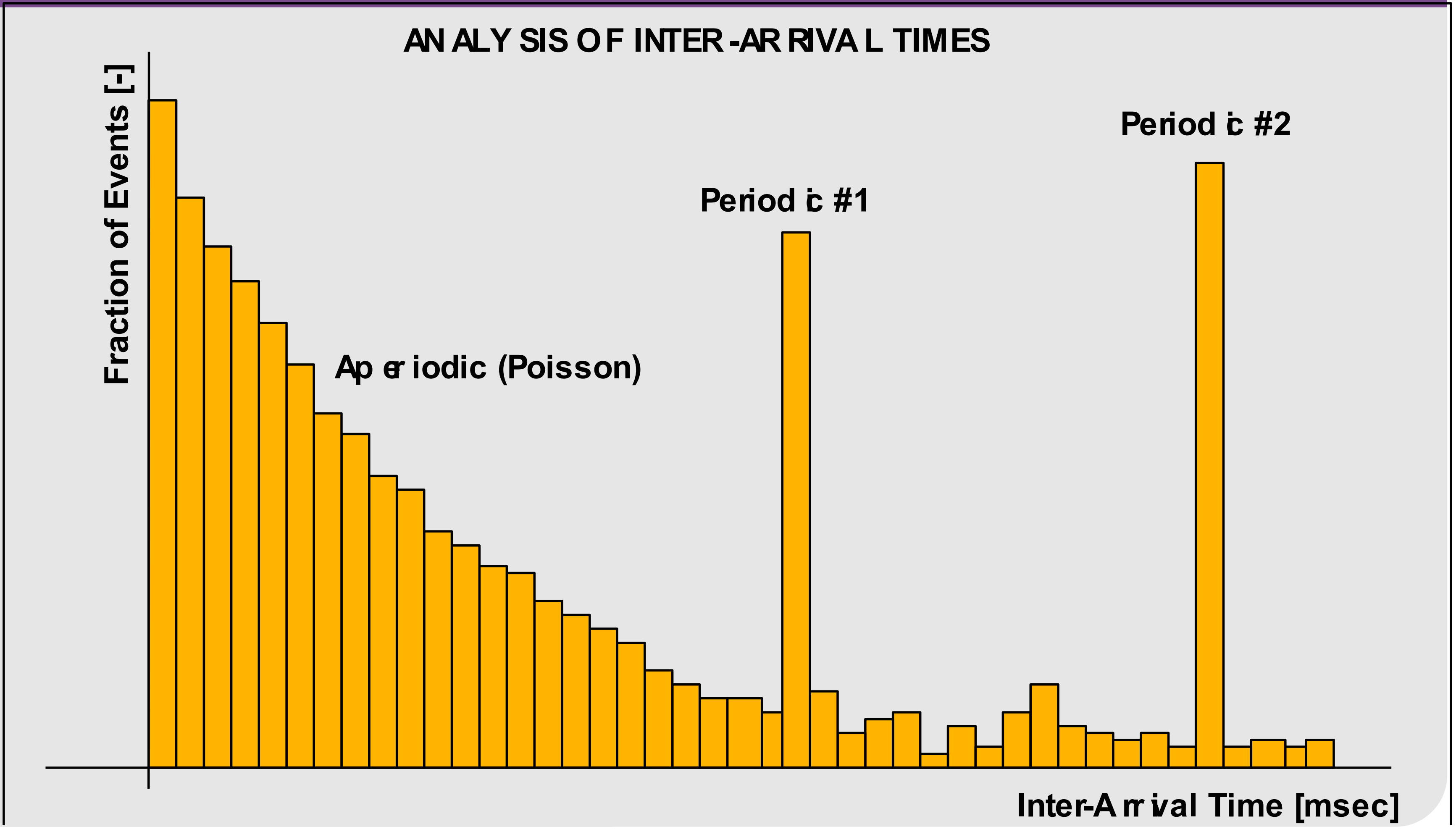
Interarrival Times



Queuing Theory Concepts (2)

- Interarrival times are distributed statistically in time, and the statistical properties of this distribution are very important if we wish to model and analyse the behaviour of a specific queue type.
- To model interarrival times, we require knowledge of their statistical properties.
- This is usually determined collection of large amounts of statistical data, and then analysing the data.
- Broadly we can divide interarrival times into *periodic* and *aperiodic*.
- Aperiodic interarrival times are random, and behave as a random process.

Interarrival Times Poisson + Periodic



Poisson Process

- Very common random process in nature.
- Usually human behaviours follow Poisson statistics – frequency of phone calls, database accesses, web server accesses.
- Random failures in equipment also exhibit Poisson properties.
- The rate of the process is λ (lambda). P is probability.

$$P_n(t) \quad = \quad P(\# \quad Arrivals \quad = \quad n \quad at \quad time \quad t)$$

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$$

MONASH University
Information Technology

www.infotech.monash.edu

Poisson Process Example (Robertazzi)

- A telephone exchange receives on average 100 calls per minute, Poisson distributed in time.
- What is the probability that no calls are received in an interval of five seconds?
- Tutorial – do some other examples

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} = \frac{(100x1/12)^0}{0!} e^{-100x1/12} = 0.00024$$

Other Poisson Properties

- **The mean of of a Poisson process can be shown to be:**

$$<n> = \lambda t$$

- **The variance of a Poisson process can be shown to be:**

$$\sigma_n^2 = \lambda t$$

- **The *memoryless* (Markov) property: *the distribution of time until the next event is the same no matter what point in time we are at (Robertazzi).***
- Example: **Trains arrive at a station with a Poisson distributed mean time between trains of 20 minutes. The last train arrived 19 minutes ago. When will be next train arrive?**
- **Is the correct answer 1 minute?**

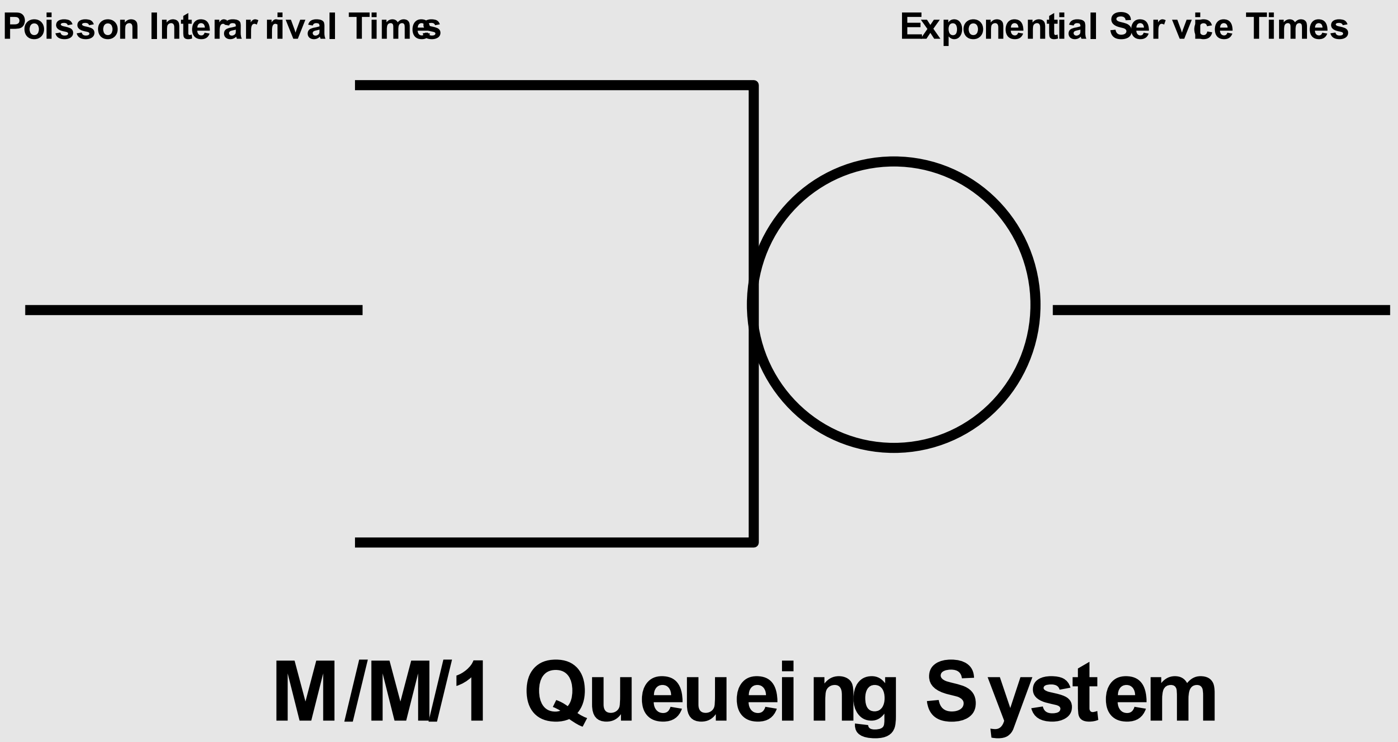
Poisson and Service Times

- Until now we have looked at Poisson behaviour in arrival times.
- Service time in a queue is the time it takes for the task being queued for to be executed and completed.
- The duration of telephone calls is an example of service time.
- How are service times distributed statistically?
- For many systems, services times are also Poisson, where μ is mean service rate:

$$P_n(t) = \frac{(\mu t)^n}{n!} e^{-\mu t}$$

Markovian Systems – M/M/1 Queueing System

- Markovian arrival times, Markovian service times, single queue.



Properties of M/M/1 Queues

- Whether we look at a bunch of villagers queueing for a well, a telephone system, or a switching device, we will be interested in how long the queue will be, and what waiting time is spent in the queue, as the load on the system varies.
- A light load is where $\lambda \rightarrow 0$ and a heavy load where $\lambda \rightarrow \mu$ (*maximum)* for the system of interest.
- We define utilisation $U = \rho = \lambda / \mu$; utilisation is a measure of how busy the queueing system is, by comparing the arrival rate with the service rate.
- If the arrival rate is 0, $U = 0$, if the arrival rate is as big as the service rate, $U = 1$.

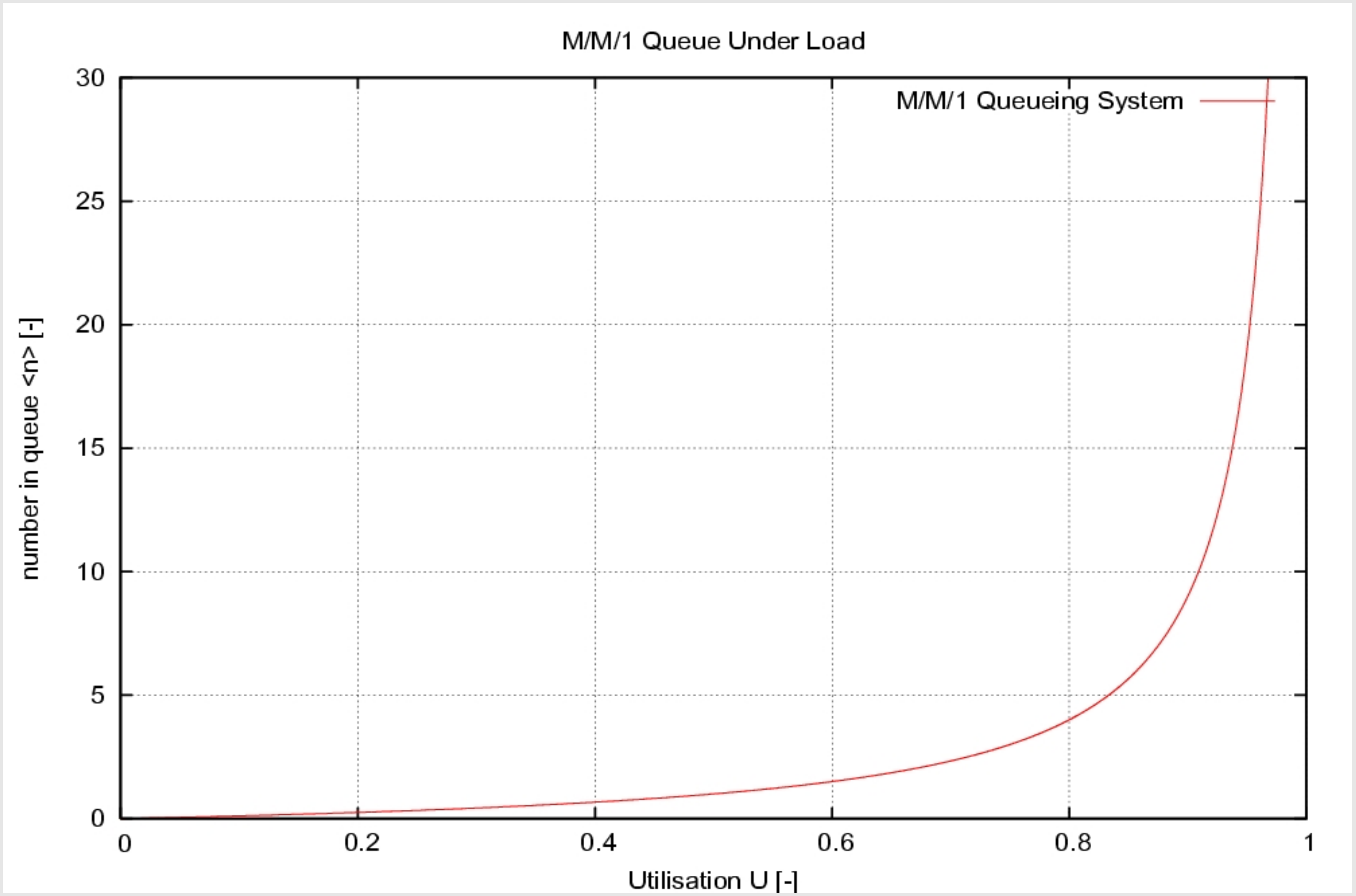
What Happens as ρ → 1?

- As we increase the load on an M/M/1 queue, an interesting thing happens.
- The average number in the queue can be shown to be:

$$< n > \; = \frac{\rho}{1-\rho}$$

- As utilisation approaches 1, the queue saturates, and the lower half of the fraction tends to zero, driving the length of the queue to infinity.
- Saturated queues therefore experience extremely long delay times, as tasks pile up waiting to be serviced.

What Happens as $\rho \to 1$?



Little's Law

- **Little showed that the mean number of customers in a queue, the arrival rate, and the mean waiting time are related by:**

  $<n> = \lambda <\tau>$

- ***Where <τ> is* the mean waiting time.**
- **We can then determine <τ> *to be:***

$$<\tau> \quad = \quad \frac{1/\mu}{(1-\rho)}$$

Reading and References

- Extensive research and literature exists. Recommended reading on queueing:

1. L. Kleinrock, *Queuing Systems, Volumes I & II*. New York: Wiley, 1976
2. Lazowska E.D. et al, Quantitative System Performance, http://www.cs.washington.edu/homes/lazowska/qsp/
3. Thomas G. Robertazzi, Computer Networks and Systems, Springer-Verlag, 1990
4. Myron Hlynka's Queueing Theory Page URL: http://www2.uwindsor.ca/~hlynka/queue.html
5. Myron Hlynka's Queueing Definitions URL: http://www2.uwindsor.ca/~hlynka/qdef.html
6. Myron Hlynka's Queueing Humor URL: http://www2.uwindsor.ca/~hlynka/qfun.html