University of Portsmouth
BSc (Hons) Computer Science
Third Year

Distributed Systems And Security (DSSEC)

M30225

January 2026 - June 2026

20 Credits

Thomas Boxall
thomas.boxall1@myport.ac.uk

Contents

Part I

Distributed Systems

Page 1

Lecture - Module Intro & Overview of DS

2026-02-03 09:00 Amanda

1.1 Introduction to this Module

This is a module of two parts:

- Distributed Systems taught by Dr. Amanda Peart, first 8 weeks with 1 lecture (1 hour) and 1 seminar (2 hours) per week. Exam to cover this content worth 60% which will be held during March, before the Easter break.
- Security taught by Dr. Fahad Ahmad, which will have a different schedule. The exam to cover this content will be during the summer exam period and worth 40% of the overall module.

For Distributed Systems - there are two books recommended, one by Coulouris which is available in the UoP E-Library and on GitHub; and one by Tanenbaum. Both are relevant in places to supplement lecture material, regardless of their publish date.

There's some more more information on the content to be covered in this Module available on the Moodle page.

1.2 What is a Distributed System?

A *Distributed System* is the interaction between two or more computing based devices which are connected in some fashion. Historically, this would have been seen as two Laptops, or PCs, then more recently as mobile phones, and even more recently as Meta Glasses (etc). This is an ever evolving field, however the two primary issues remain the same:

- How do the devices communicate? For example through which networking protocols and methods
- How do the devices interact? For example through distributed software

Definitions

Distributed System "A distributed system is a collection of independent computers that appear to the users of the system as a single computer" - Tanenbaum.

Distributed System "A system in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing" - Coulouris

1.3 Why use Distributed Systems?

In short, we want to do things faster. Expectations for technology are increasing - the users demand the service to be instant, the webpage to load immediately, and to never see the loading wheel of

death.

There are three methods which can generally be employed to to do things 'faster': working harder; working smarter; and getting help.

1.3.1 Working Harder

Working Harder is the idea of putting more effort into the task, in the hopes of completing it faster or better. For example, the night before exams cramming as much as possible. It can also apply to the thought of finding shortcuts to achieve given tasks. For example Michaelangelo carrying his lunch up the scaffold with him so he didn't have to stop painting, or a desperate student taking their lunch to the library to avoid that 2 hour side quest to Greggs.

In computing - we can see working harder through having higher preforming hardware.

However working harder alone often isn't enough.

1.3.2 Working Smarter

Working Smarter is the idea of finding a way to reduce the amount of work needed to accomplish something. Taking a task and making it more efficient or smarter has a positive side effect, in that you get more out of it with less effort. For example, McDonalds have a streamlined kitchen, or a racing line used by Formula One drivers to minimise the distance to cover.

In computing - we can see working smarter through more efficient algorithms.

Although, even if we manage to work harder and more efficiently, that still may not be enough.

1.3.3 Get Help

Getting Help is the idea of more computers, computational devices, etc, being involved in the process to increase capacity. For example, a group of roadworkers digging a hole in the road is more efficient than one person on their own.

In computing - we can see getting help through distributed or parallel processing.

However, there are negative cases of getting help too. For example, some committees spend longer discussing and thinking than doing, where an individual or smaller committee may make the decision in a quicker and less painful method. In saying this, getting help is good, but you must be aware of the risks associated.

1.4 Where are Distributed Systems Used?

Distributed systems can be seen in lots of places. They're commonly embedded in technology:

- Network File System (NFS)
- SSH / telnet
- Web - retrieving a page or used in CGI Scripts
- Network Printers
- ATM (Cash Machine)

Alternatively in more specialist technology:

- Distributed Shared Memory (DSM)
- Distributed Objects (COBRA or Java RMI)

- Process Migration (Condor)
- Replication (NIS)
- Network Video & Audio

Most of the above systems / applications communicate in some way with other systems, applications or entities. This means they communicate through a network. There are some requirements for the network. (Flashbacks to Networks L4 module).

1.4.1 Networks Requirements

In a traditional layered protocol stack, we may use the OSI or TCP/IP protocol stacks. These have defined layers and defined functions:

- Data link layer - handles the bits on the wire
- Packet switching
- Addressing and routing
- Reliable data streams
- Remote Procedure Calls (RPC) & other high level protocols (SNMP, HTTP, etc)

However in the land of distributed systems, the layered architecture isn't so present; where it's more common for protocols, APIs and services to be mixed and matched as required. Some layers do exist, however:

- RPC, Time & directory & security, File Server
- Storage, Directory, Replicated FS.

1.5 Structure of Distributed Systems

A distributed system can be considered to have three levels: Fundamentals, Distributed OS, and Application.

The *Fundamental Elements* level is the lowest level. This is concerned with the individual processes, threads, and concurrency controls. It also handles the synchronisation between different devices, and the Remote Procedure Calls (RPC). The lowest level also deals with naming the different components for clarity and ease of access, caching and finally network protocols. Many of the topics listed here were covered during the Operating Systems and Internetworking module at Level 5.

The *Distributed Operating Systems* level sits in the middle. This is concerned with scheduling, transactions, replication, fault tolerance, persistence and high availability. Many of these elements are to do with ensuring the multitude of devices in the distributed system are all using and referencing the same same data.

The *Application* level is the highest level. It handles any distributed shared memory, distributed objects, manages security and protected environments.

1.6 Issues & Considerations in Distributed Systems

Within a distributed system, there can be a need for *costly communications*. Most commonly this is seen in a setting where Symmetric Multiprocessing (SMP) is not available, leading to multiple discrete but connected nodes needed. The bandwidth and latencies of SMP is unparalleled to that of multiple nodes - causing performance degradation.

Unreliable Communications are a significant issue between distributed systems. If the connection

is unreliable or unavailable altogether, messages may be lost or corrupted which means a remote computer may not be usable.

Distributed Systems need to have resiliency built in; a system should be able to continue it's operation after the *independent failure* of a node, preferably without the user noticing any performance issues unless the failure is significant.

The communication between nodes can also be problematic, where *insecure communications* could be exposed leading to unauthorised access, which potentially may be malicious.

1.7 Design Issues

When designing a distributed system, or even a system which may need to be distributed in the future, there are a number of issues with should be considered to ensure the effective functioning of the system.

1.7.1 Naming

Names assigned to resources or objects must have global meanings. This refers to any names or identifiers used in the code or deployments. The names should be independent of the locations of the objects. There should be a name interpretation system that cna translate names in order to enable different programs to access named resources. Names used should scale and translate efficiently.

1.7.2 Access and Authorisation

Authentication should be global service across the entire distributes system; users shouldn't need to authenticate for every different component of the same application. The same functions should be usable everywhere and should have reasonable performance.

1.7.3 Communications

The performance and reliability of the communication techniques used for the distributed system are critical to the performance of the system. This means that the data communicated within the system should always be up to date, so definitely not yesterday's data presented as though it's today's.

1.7.4 Software Structure

Open software is achieved through the design and construction of software components with well defined interfaces, such as APIs. Through an open system, more people can improve and access the system, however it does case security issues through the number of people accessing the system.

Data abstraction within the software is important, as well asa designing a framework that can work with existing and new services without duplicating the existing services.

1.7.5 Resource Management

While designing the distributed system, consideration must be given to the deployment. Optimisation of one component of the processing or communication resources within the network can simply move the bottleneck to another component in the system. Load balancing can be employed to move the workload between different systems.

1.7.6 Consistency Maintenance

Within a distributed system there may be multiple nodes performing the same function or needing to access the same data; herein lies issues with consistency. Consistency is concerned with:

- Data - where multiple nodes attempt to access a single data item, will they both get the same item?
- Replication of data - fault tolerance in this
- Cache - performance improvements, FT, availability
- Failure - is there a master / slave architecture setup
- Consistent time
- User Interface - ensuring ease of use and smoothness to users

1.7.7 Security

Authentication ensures users have the right access to resources. Access Control provides services to give different users different levels of access. Auditing keeps logs of all actions taken.

Page 2

Lecture - Communication Models

2026-02-10 09:00 Amanda

Distributed Systems communication models define how different processes and nodes (systems) interact with each other. The models influence the design of the system, efficiency and reliability. One model used may feed communications to another model, and therefore a different system.

There are a number of different models which will be explored in this lecture. However this is not an exhaustive list - as that would fill 8 weeks of lectures, not just a 1 hour lecture!

We will never negate all the issues with these communication models, however we can try to improve them. The networks connecting devices together are improving which is inherently improving the models.

2.1 Message Passing

Message Passing works by processes sending and receiving messages over a network. This is commonly used in distributed computing for inter-process communication.

Message Passing can work asynchronously or synchronously to exchange messages. It works both ways so it can best fit the message contents and what needs to happen with the data on receipt. Messages must be serialised before transmission and deserialized on receipt. Messages passing is susceptible to network failures, so some thought has to be given as to how to overcome this when designing the system. Message passing is a form of direct communication.

Typically, message passing is seen where a process needs to communicate with another process. Through the first process passing the second a message - the second process then has a copy of the data. From this arises a need to keep the data consistent across the network.

Examples of applications of message passing:

- Web Sockets (TCP / UDP)
- Remote Procedure Calls (RPCs)
- Message Queue Systems (Rabbit MQ, Kafka)
- REST APIs

2.2 Shared Memory

Shared Memory is a model in which multiple processes / nodes read from and write to a shared memory space. This model is more common in multi-threaded systems but can be used in distributed systems where it becomes the *Distributed Shared Model* (DSM). With the uprising in cloud computing, the memory can now geographically be anywhere.

The Shared Memory model doesn't have a need for message exchange, rather the systems just directly interface with the shared memory. However it does require some form of synchronisation - often seen with Locks or Semaphores (as seen in L5 OSINT).

To the user, shared memory is seen as a single memory location when in reality - this is split across multiple different nodes.

Examples of applications of shared memory:

- Distributed Shared Memory
- Distributed File Systems (Google Drive, Hadoop HDFS)

An issue with shared memory is the need for ensuring that the data is saved on a sensible node. What this means is that if Node 1 is asking Node 2 for some data, where in reality only Node 1 needs this data - then it would be most sensible to store the data on Node 1. This should also be extended to ensure that the data is stored on a node which is geographically close to the users.

2.3 Remote Procedure Calls

Remote Procedure Calls (RPCs) allow a process to invoke a function on a remote system as if it were a local function call. They abstract the details of the communication on the network.

RPCs work through the client sending the request to the server which processes the request and then responds to the client with it's response. They can be blocking (synchronous), or non-blocking (asynchronous). The data used in an RPC requires serialisation before it can be transmitted on the network, which is commonly achieved using protocol buffers or JSON.

Within a RPC system - there has to be a clear and consistent naming convention - something which can be understood by both the client and server. On the server side, there may be a stub which can translate received names into local names, alternatively to decide if the data can be request can be processed or not.

Examples of applications of RPCs:

- Google RPC (gRPC)
- Apache Thrift
- Java Remote Method Invocation (Java RMI) - which was one of the big features that made Java explode in popularity

2.4 Publish/Subscribe Model

The Publish/Subscribe (Pub/Sub) model is an event-driven communication method. Publishers broadcast messages without knowing what subscribers (if any) are listening for them.

This is a decoupled architecture, where there is no direct connection between the sender and receiver. Communications are handled asynchronously. Pub/Sub is seen in real-time streaming and event-driven architectures.

Examples of applications of Pub/Sub:

- Apache Kafka
- Google Cloud Pub/Sub
- MQTT

2.5 Data Streaming Model

The data streaming model handles processes continuously sending and receiving data in real time. This is ideal for applications that need low latency data transmissions.

Data streaming often involves continuous data flow working in a push model, where a process continuously pushes data at the receiver who has to deal with the flow in some way. There may be a broker of sorts in the middle of the flow, cleaning the data or merging it with other data streams. This model is suitable for real-time analytics and monitoring systems and can commonly be seen working with Pub/Sub systems.

Due to the relentless data pushing aspect of the model - there is a need for a lot of bandwidth to handle the traffic and some decent infrastructure for the system to run on. We strive for there to be zero latency in this system in case we are looking to make critical decisions on realtime data. There may also be some storage for archived data, which is separate to working storage.

Examples of applications of Data Streaming:

- Apache Kafka
- Apache Flink
- Spark Streaming

2.6 Peer-to-Peer

Peer-to-Peer (P2P) is a model in which all nodes in the system act as both a client and server. They share resources and data without a central authority commanding and conducting the way they work.

P2P is a decentralised architecture, it is easier to bring in nodes and ensure replication between them. It is highly scalable but may have inconsistency issues between nodes. P2P is used in blockchain, file sharing and decentralised applications.

In a way, utilising P2P is almost like utilising shared memory. However there is a risk that if one node goes down and there is no suitable data replication in place, the entire system could loose access to the data that node is holding.

In the early days of P2P systems, back to the Limewire era, there were a number of issues with the P2P software utilising too much of the PCs resources, making the system unusable for the user.

Examples of applications of P2P:

- Bit Torrent
- Blockchain
- Skype

2.7 Tuple Space Model

The Tuple Space Model is an example of coordination based communication, where the processes interact through a shared tuple space. The processes read, write and take data tuples instead of direct message exchange.

This model is based on the Linda Coordination Models. It is decoupled in time and space as the processes don't need to be online simultaneously - however does require some coordination. It is used in distributed coordination and parallel computing. This can be more efficient, however there are complexities with managing the tuples effectively.

Examples of applications of Tuple Space:

- JavaScript
- Apache ZooKeeper

- Tibco Rendezvous

2.8 Cons With These Systems

There are of course cons with using these communication models:

- Message Passing - requires message handling
- Shared Memory - synchronisation overhead
- RPC - can be blocking to other processes / applications
- Pub/Sub - latency in event delivery
- Data Stream - complex implementation
- P2P - data consistency issues
- Tuple Space - complexity in managing tuples

Freely Given, Highly Questionable

These notes are a byproduct of my learning process - mistakes, bad jokes and snark included. If you take an error of mine as gospel and ruin your perfect 1st, that is a *you* problem boo. I am a student, not a prophet, use a textbook if you want certainty.

Yes, they have been typeset in LaTeX; I'd rather typeset in stone with a blunt chisel than use Word Online for another minute. If the formatting looks obsessive, it's because it was either this or a breakdown. You can find the source on my GitHub.

Thomas Boxall is a Computer Science Student studying at the University of Portsmouth, UK. In all honesty, he should probably be revising or doing some work towards his dissertation, not fondling the template of his notes again. You can find Thomas online at thomasboxall.net