

Final Submission Socket Programming and Concurrency

by Rhys Davies

Submission date: 10-May-2025 09:45AM (UTC+0100)

Submission ID: 258680273

File name:

24078_Rhys_Davies_Final_Submission_Socket_Programming_and_Concurrency_664976_988234313.docx
(348.34K)

Word count: 3005

Character count: 17996

Socket Programming and Concurrency Report

Understanding Key Network Protocols and Their Impact on Application Design

Model Code: ACCA6028

Rhys Davies | 2008896

May 2025

Contents

Socket Programming and Concurrency Report	1
Introduction	2
Transmission Control Protocol (TCP) and Data Delivery	2
1.1 Overview of the TCP Operation	2
1.2 Connection Establishment	3
1.3 Data Transfer	4
1.4 Flow Control	4
1.5 Congestion Control	5
1.6 Connection Termination	5
2. Impact of Maximum Transmission Unit (MTU) on Data Transfer	6
2.1 What is MTU?	6
2.2 Protocol Data Units (PDU) and Open Systems Interconnection (OSI) Model	6
2.3 IP Fragmentation	7
3. Synchronicity and Asynchronicity in Network Protocols	7
3.1 Synchronous Communication	7
3.2 Asynchronous Communication	8
3.3 Input/Output (I/O) Dispatch Models	8
4. Interoperability in Networked Applications	8
4.1 What is Interoperability?	8
4.2 Achieving Interoperability	9
5. Conclusion	10
6. References	11
Figure 1 - Two Generals' Problem	4
Figure 2: TCP Connection Termination	5
Figure 3 - ISO Model	6

Introduction

In the current age, the dependability, performance, and mix-and-match capability of Internet applications is vital to modern digital existence. This report discusses socket programming and concurrency, particularly of the Transmission Control Protocol (TCP) and how the TCP is responsible for a robust data communication as supported in this report. It discusses the performance and reliability effects of techniques such as flow control, congestion avoidance, and packet fragmentation. The report then goes on to compare synchronous and asynchronous communication models to give a better understanding and reasoning behind application design decisions. Lastly it examines how Interoperability has a key role in enabling the integration between different systems, and the role of standard protocols and APIs in application design.

Transmission Control Protocol (TCP) and Data Delivery

1.1 Overview of the TCP Operation

The Transmission Control Protocol (TCP) is a core protocol within the Internet Protocol (IP) suite, providing reliable, ordered, and error-checked delivery of data between applications across a network. In contrast to connectionless protocols such as User Datagram Protocol (UDP), TCP establishes a connection-oriented communication channel, ensuring that data is transmitted accurately and in sequence, especially over unreliable networks. [1]

1.1.1 Purpose and Functionality

In networking, data often needs to be transmitted across various devices and networks where packet loss, delays, and reordering can occur. TCP mitigates these challenges by using the following mechanisms:

- Connection establishment and termination which creates a reliable communication link between the sender and receiver.
- Sequencing and acknowledgement to ensure that data arrives in the correct order and that packets lost during the process are retransmitted.
- Error detection and correction through checksums, preventing corrupted data from being accepted by the receiver.
- Flow control to manage the rate of data transmission based on the receiver's capacity.
- Congestion control to prevent overwhelming the network which ensures an efficient and stable communication between the sender and receiver.

These features make TCP ideal for applications where data integrity and reliability are a priority, such as web browsing (HTTP/HTTPS), email (SMTP), file transfers (FTP), and remote access (SSH). [1]

1.2 Connection Establishment

1.2.1 The Two Generals' Problem

A common analogy used to describe the challenge of communication over unreliable channels is the 'Two Generals' Problem' as seen in Figure 1 - Two Generals' Problem. There are two generals, and they each have an army at their disposal in different locations and they want to coordinate an attack on a town belonging to an enemy but there's no way they can communicate with each other or agree on when to attack, except through messengers, messengers need to go through the enemy's territory and there is a possibility that the messengers get intercepted. Some of the issues with this problem include:

- General 1 sends a message to General 2 proposing a time for the attack. However, the message may not reach General 2.
- Even if the message reaches General 2, how can General 1 be certain it was received?
- General 2 might send an acknowledgement back to General 1, but this acknowledgement could also be lost.

To be fully assured, each general would need to send an acknowledgement of the acknowledgement, and so on, leading to an infinite loop. TCP tries to work around that however with things like acknowledgements or retransmissions, but you can never eliminate the uncertainty of the Two Generals' Problem. [2]

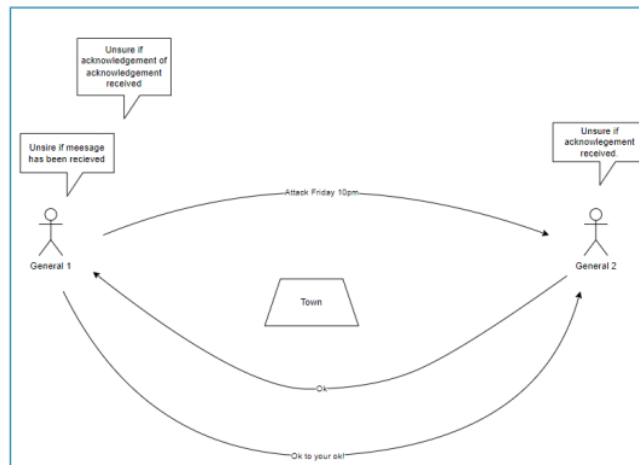


Figure 1 - Two Generals' Problem

1.2.2 The Three-way Handshake

A three-way handshake is a critical process in TCP to ensure a reliable connection between a sender and receiver. The steps are as follows:

- **Synchronise (SYN):** The Client sends a SYN message to the server to initiate a connection. The packet contains a randomly sequenced number that is used for tracking during data transmission.
- **Synchronise-Acknowledge (SYN-ACK):** The server responds with a SYN-ACK message communicating it is ready to accept data. This acknowledges the Client's SYN request and adds its own randomly sequenced number.
- **Acknowledge (ACK):** The Client replies with an ACK message which confirms the connection and synchronisation of the two devices. The connection is now established, and data can now be transmitted. [3]

1.3 Data Transfer

Once the connection is established, TCP provides reliable, ordered, and error-checked delivery of data. The data is split into smaller segments, with their individual sequence numbers, allowing the receiver to reassemble the segments in the correct order on the other end. TCP uses acknowledgements to confirm receipt of segments and retransmission timers to resend any lost or corrupted data, ensuring that the data on the other end is received complete and accurately. [1]

1.4 Flow Control

TCP manages the rate of data transmission using a flow control mechanism based on the receiver's available buffer space which is designed to prevent the sender from overwhelming the receiver. This is achieved using a 'sliding window protocol', which allows multiple packets to be sent before requiring an acknowledgement. The window size is dynamically adjusted based on the receiver's advertised window size and current network conditions. [4]

1.5 Congestion Control

In addition to flow control, TCP manages congestion across the network through several algorithms, including slow start, congestion avoidance, fast retransmit, and fast recovery.

- Slow Start: TCP begins transmission cautiously, gradually increasing the transmission rate to probe the network's capacity.
- Congestion Avoidance: After reaching a threshold, TCP increases its transmission rate more slowly to avoid causing congestion. [5]
- Fast Retransmit and Fast Recovery: These mechanisms quickly detect packet loss without waiting for a timeout and adjust the congestion window to recover efficiently. [6]

1.6 Connection Termination

Similar to the three-way handshake, TCP connections are closed using a four-way handshake to ensure all data is successfully transmitted and acknowledged as seen in Figure 2: TCP Connection Termination.

- The sender transmits a FIN (Finish) packet to the receiver to initiate termination.
- The receiver acknowledges the FIN with an ACK.
- The server then sends its own FIN packet once it has finished transmitting remaining data.
- The Client responds with an ACK to complete the process. This method ensures that both sides can independently finish sending and acknowledging any outstanding data before the connection is fully closed.

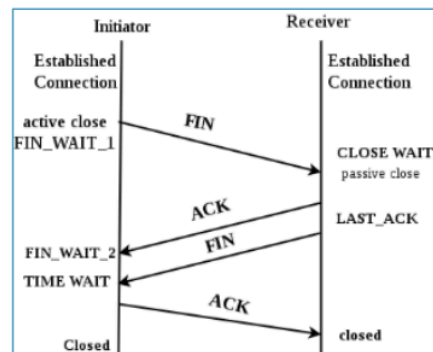


Figure 2: TCP Connection Termination

<https://www.geeksforgeeks.org/tcp-connection-termination/>

2. Impact of Maximum Transmission Unit (MTU) on Data Transfer

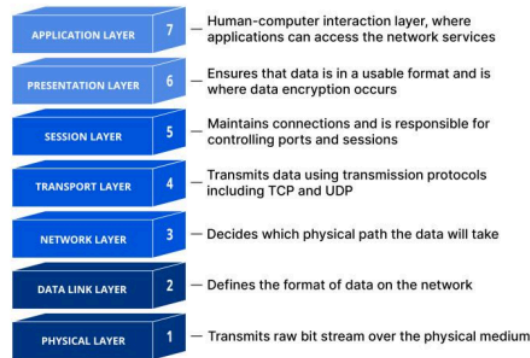


Figure 3 - ISO Model

(<https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/>)

2.1 What is MTU?

The Maximum Transmission Unit is a parameter that specifies the maximum size of a packet which can be sent on any given network. Various network types such as Ethernet or Wi-Fi set an MTU size, which in turn determines how much data can be put into each packet. Ethernet networks have a commonly used MTU value of 1500 bytes, while Wi-Fi networks generally have an MTU value slightly lower due to more wireless overhead.[4]

Setting the correct MTU is critically important for maintaining a network's efficiency and performance. If necessary, large packets can be sent rather than many smaller ones to avoid the overhead of transmitting so many small pieces of data. That's because too many small packets might cause congestion and so congestion would reduce throughput across the network. [4]

2.2 Protocol Data Units (PDU) and Open Systems Interconnection (OSI) Model

Each layer of the OSI model (as seen in Figure 3 - ISO Model) handles data using specific structures known as Protocol Data Units (PDUs). At the Data Link Layer, the PDU is called a frame. At the Network Layer, data breaks on down into packets. At the Transport Layer (where protocols such as TCP and UDP operate), it becomes a segment. The size of each PDU is controlled by the MTU of the network so data is split into smaller segments for efficient transfer across the network.

Developers need to make sure how the connection is handled in different layers. Only then can we rely on them to write efficient and reliable applications. As an example, the transmission of large documents may force developers to take data in chunks; otherwise, they could incur both unnecessary division of the data and strain on a network. [5]

2.3 IP Fragmentation

If a packet is larger than the MTU limit, it must be divided into smaller pieces with each fragment assigned a different header. The packet can be fragmented by the Client or a router in IPv4. Assume each fragment as its own packet and add on header information so that the Server can reconstruct the original on the far side. Though it solves the limitations of the MTU, this adds extra overhead to the system especially if packets are lost. This is Simplification. However, this is its hidden penalty since it adds overhead to our system. It becomes even more burdensome if packets are lost and must be retried from the beginning.

To get around fragmentation, one can employ a technique known as Path MTU Discovery (PMTUD), where the Client finds out the lowest MTU along the path to the Server and adjusts packet size. Routers in IPv6 are not allowed to fragment packets so the Client should ensure that they are an appropriate size before sending them off. [6]

3. Synchronicity and Asynchronicity in Network Protocols

3.1 Synchronous Communication

When there is synchronous communication, two or more devices exchange data in real time. It requires both the Client and the server be active and responsive at the same time. A traditional phone call is an example of synchronous communication when one side or the other of the communication channel stops responding, the conversation cannot go on. In networking, synchronous models are typically utilised when immediate effect is required. Protocols like HTTP/1.1 can be synchronously operated - a Client sends a request and must wait for the server to completely respond before proceeding. Synchronous designs are often more easily implemented, because communication follows a predictable request-response pattern. However, inefficiencies arise when the network is slow or when high latency is involved. Idle time and wasted resources follow. [7]

3.2 Asynchronous Communication

Through asynchronous communication, devices and processes can send messages without needing immediate responses. The Client can go on working, while the server processes the message on its own. Email is an example of asynchronous communication. An email may be sent without needing an immediate response. Message queues adopted by REST APIs allow applications to scale efficiently. They do not block operations while they await replies. Many Web servers, like Node.js, are built upon an asynchronous and event-driven model that enables them to handle large numbers of simultaneous connections without suffering from thread throttling. [10]

3.3 Input/Output (I/O) Dispatch Models

I/O dispatch models stipulate how systems handle multiple input and output operations. In a synchronous blocking model, applications wait for I/O to complete before continuing with other tasks. A commonly used analogy is to compare the operation to a waiter in a restaurant who serves one table at a time. The approach is simple, however it's not ineffective in its scale. Asynchronous non-blocking models, using event-driven programming, allow applications to keep working while they are waiting for I/O operations to push through. This is in some ways akin to a waiter that she serves many tables by switching between them. To get great crowd, because an application doesn't have to wait for a single job to complete before proceeding with another. [11]

A thread-per-connection approach assigns each connection to its own thread. Like a waiter who serves each table independently, this approach simplifies the logic since each thread handles one task, but it becomes inefficient with many connections due to the overhead of managing many threads. This is like having too many waiters serving only one table, which can cause unnecessary complexity and resource consumption.

Event loops or the reactor pattern use a single thread to manage events, making them ideal for high-concurrency systems, much like a single waiter who listens for requests from all tables and quickly addresses the next available task. This model is efficient because it can handle numerous tasks concurrently without requiring multiple threads, but it can also be more complex to implement. [11]

4. Interoperability in Networked Applications

4.1 What is Interoperability?

Different systems, applications, and devices may use entirely different technologies, architectures or standards alone; when they work together Interoperability is the quality that lets them work together smoothly with no loss of function. In networked applications, it means that the systems as

such on different platforms (such as operating systems, hardware configurations, or programming languages) can communicate effectively. This is often necessary for making possible cooperation between organizations with different backgrounds. It's also important when you're trying to maintain large heterogeneous systems containing components from many vendors running different operating systems or based on altogether dissimilar technical foundations.

A Windows user should be able to access resources located on a Linux server, just as easily as do apps from Java communicate with services written in Python. Interoperability removes the need for a deep understanding of the other side's architecture, making it easy to move data across different environments. [12]

4.2 Achieving Interoperability

Interoperability depends on standardised protocols, data formats, and interfaces that assure all parties have an understanding of how to exchange this information. The most basic way is to use common communication protocols and data forms. For web communication, protocols like HTTP and HTTPS are indispensable. These protocols allow web browsers, servers and others to exchange data no matter what their previous systems may have been. For example, a website served from an Apache server can serve pages to either the windows or mac operating system because both the browser and server understand HTTP.

JavaScript Object Notation (JSON) and Extensible Markup Language (XML) are widely used ICE formats for exchanging structured data. Both formats are commonly employed in APIs so that different applications can parse or process the data, whatever environment in which they were developed notwithstanding. APIs are necessary for achieving interoperability. They provide standardised interfaces so that applications can communicate. RESTful APIs are often used for web-based services, and they are built according to a set of principles. These principles, which the architects call simple, stateless, and scalable, make the RESTful API usable in many different environments.

For example, a weather service might offer a RESTful API that returns JSON data which can then be used by a program. Using standard protocols, data formats, and APIs, developers can ensure that one application is able to cooperate with other systems even if those systems are built with different technologies. Standardisation is not only a vital part of getting interoperability built into today's applications but also makes it possible to build applications that can last well into the future as new technologies and platforms emerge. [12]

5. Conclusion

Socket programming and concurrency are essential for building networked applications that are both reliable and efficient. This report has looked at how data integrity and order are maintained by TCP, with features such as sequencing acknowledgements, flow control and congestion management which are important to achieve reliable communication.

The report also discussed the effect of Maximum Transmission Unit (MTU) on data transfer and packet sizes. Programmers need to be mindful of this to improve application throughput and to minimise inefficiencies.

Furthermore, synchronous and asynchronous communication methods were compared and discussions around the importance of choosing the right approach that meet the requirements of the application. Systems that have high concurrency volumes in their communications, especially those using asynchronous models, offer some scalability and responsiveness. The need for interoperability is something that never changes in modern application development. By using standardised protocols, data formats, and APIs developers can help ensure seamless communications between systems built on different technologies.

6. References

- [1] TechTarget, "Transmission Control Protocol (TCP)," June 2024. [Online]. Available: <https://www.techtarget.com/searchnetworking/definition/TCP>. [Accessed 27 April 2025].
- [2] Medium.com, "Two Generals' Problem and Idempotency," 26 September 2024. [Online]. Available: <https://medium.com/@ali.gelenler/two-generals-problem-and-idempotency-4d28f4b07694>. [Accessed 27 April 2025].
- [3] Codecademy.com, "The TCP Three-Way Handshake Explained," 10 February 2023. [Online]. Available: <https://www.codecademy.com/resources/blog/what-is-tcp/>. [Accessed 27 April 2025].
- [4] GeeksforGeeks, "Sliding Window Protocol," 25 July 2024. [Online]. Available: <https://www.geeksforgeeks.org/sliding-window-protocol-set-1/>. [Accessed 27 April 2025].
- [5] GeeksforGeeks, "TCP Congestion Control," 28 December 2024. [Online]. Available: <https://www.geeksforgeeks.org/tcp-congestion-control/>. [Accessed 27 April 2025].
- [6] SystemsApproach, "6.3 TCP Congestion Control," [Online]. Available: <https://book.systemsapproach.org/congestion/tcpcc.html>. [Accessed 27 April 2025].
- [7] ESEYE Documentation, "About the maximum transmission unit (MTU)," [Online]. Available: [https://docs.eseye.com/Content/Connectivity/MTU.htm#:~:text=The%20maximum%20transmission%20unit%20\(MTU\)%20is%20a,over%20a%20network%20in%20a%20single%20transmission.&text=For%20Ethernet%2Dbased%20networks%20\(including%20wired%20Ethernet%20and,the%20standa](https://docs.eseye.com/Content/Connectivity/MTU.htm#:~:text=The%20maximum%20transmission%20unit%20(MTU)%20is%20a,over%20a%20network%20in%20a%20single%20transmission.&text=For%20Ethernet%2Dbased%20networks%20(including%20wired%20Ethernet%20and,the%20standa). [Accessed 27 April 2025].
- [8] TechTarget, "What is a protocol data unit (PDU)?," [Online]. Available: <https://www.techtarget.com/searchnetworking/definition/protocol-data-unit-PDU>. [Accessed 27 April 2025].
- [9] Cloudflare, "Broken packets: IP fragmentation is flawed," 18 August 2017. [Online]. Available: Each layer of the OSI model handles data using specific structures known as Protocol Data Units (PDUs). In the Data Link Layer, the PDU is referred to as a frame. When data passes through the Network Layer, it becomes a packet, and at the Transport Layer,. [Accessed 2025 April 27].

- [10] GeeksForGeeks, "Difference Between Synchronous and Asynchronous Transmission," 24 July 2024. [Online]. Available: https://www.google.com/search?q=synchronous+and+asynchronous+In+network&rlz=1C1ONGR_enGB1015GB1015&oq=synchron&gs_lcrp=EgZjaHJvbWUqCAGBEEUYJxg7MgYIABBFGDkyCAGBEEUYJxg7MhUIAhAuGEMYxwEYsQMY0QMYgAQYigUyDggDEEUYJxg7GI AEGIoFMhQIBBAAGEMYRhj5ARixAxiABBiKBTIGCAUQ. [Accessed 28 April 2025].
- [11] Studying With Alex, "Non-blocking I/O and how Node uses it, in friendly terms: blocking vs async IO, CPU vs IO," 17 March 2021. [Online]. Available: <https://www.youtube.com/watch?app=desktop&v=wB9tIg209-8&t=314s>. [Accessed 28 April 2025].
- [12] T. Azad, M. H. Newton, J. Trevathan and A. Sattar, "IoT edge network interoperability," School of Information and Communication Technology, Nathan, 2025.
- [13] Cloudflare, "What is MTU (maximum transmission unit)?," [Online]. Available: <https://www.cloudflare.com/learning/network-layer/what-is-mtu/>. [Accessed 27 April 2025].

Final Submission Socket Programming and Concurrency

GRADEMARK REPORT

FINAL GRADE

70/100

GENERAL COMMENTS

A fairly good report though use of sources needs a little work. You have used a very good number of sources, though a lot of these are non-authoritative web resources. You ideally want to draw from primarily material from credible, academic journals.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13