

Final

by Scott Roberts

Submission date: 20-May-2025 01:25PM (UTC+0100)

Submission ID: 259915321

File name: 49384_Scott_Roberts_Final_664976_1420275459.docx (23.63K)

Word count: 2712

Character count: 15552

Socket Programming and Concurrency Report

Scott Roberts - 2306895

2025

Introduction.....	3
Operation of TCP.....	3
Segmentation	4
Reliability	4
Flow Control.....	4
The Three Way Handshake.....	4
Maximum Transmission Unit (MTU).....	5
Synchronicity and Asynchronicity in Network Protocols.....	6
Interoperability for Networked Applications.....	6
Conclusion	7
References	8

Introduction

This research-driven essay explores aspects of computer networking, including the Transmission Control Protocol (TCP), Maximum Transmission Unit (MTU), Synchronicity and Asynchronicity, and interoperability of networked applications. The essay will additionally consider how these factors are relevant to programmers, and how they may impact the design decisions for the development of networked applications.

In the early days of computer networking, there were many different kinds of networking technology, each with their own addressing schemes and packet sizes. Whilst it was relatively straightforward for computers on the same network to communicate, it was trickier to communicate between networks.[1] Vint Cerf and Robert Kahn described a protocol that supported the sharing of information between different networks, providing for variation in parameters such as packet size, whilst providing flow control, error checking, and more features.[2] This protocol went on to be known as TCP/IP, which was revised a number of times until it became fully functional in 1981, when version 4 of the specification was published. The specification was described in RFC 791 (Internet Protocol) [3] and RFC 793 (Transmission Control Protocol) [4]

Operation of TCP

Protocols such as User Datagram Protocol (UDP) make no guarantees of reliable data delivery.[5] According to the IETF usage guidelines for UDP, it provides only a minimal transport with “no inherent congestion control mechanisms”, and that “application designers are generally aware that UDP does not provide any reliability”, noting that UDP does not retransmit lost packets or protect against duplicate packets.[6]

This is relevant to programmers as it is important to be aware of the implications of choosing either a datagram protocol or a stream protocol. UDP is best suited for applications where performance is favoured over reliability, and where the integrity of the data is not critical, such as voice or video streaming applications. In these scenarios, UDP will deliver low latency communications while mis-ordered data or the occasional missed packet will not have a significant impact on the usage of the application. In situations where data integrity is critical, the developer must either implement their own mechanisms for ensuring integrity and reliability, or opt to use TCP. TCP is a slightly more burdensome protocol in some respects and has greater overhead, but for the reliability it provides, this is often a favourable tradeoff.

TCP embraces the natural fragility of the internet and the wide variety of networks that data can be transmitted across, and builds in safeguards that attempt to guarantee reliable data transfer. TCP offers additional functionality over UDP, including error recovery, flow control, and reliability.[7] In addition, TCP is a connection-oriented protocol, meaning it requires two machines to formally establish a connection before communication begins. This is in contrast to UDP, which is connectionless.

Segmentation

As far as the application is concerned, a TCP tunnel is a continuous stream of data. The application does not have to concern itself with segmenting the data before transmitting it across the network. Likewise, the receiving application does not need to worry about re-assembling the data, as this is all handled by TCP, which groups the data into segments. The TCP protocol itself defines how the data is segmented. This is important for application developers to know, as they do not have to implement complicated segmentation mechanisms themselves.

Reliability

One of the headers included in every TCP packet is the sequence number field, which is a 32-bit integer. At the start of a connection, the sequence number is chosen at random. Each subsequent packet will then increment the sequence number by the number of bytes sent. This effectively means that every byte of data sent is numbered. This has a number of benefits, namely:

- If a packet is not acknowledged after a timeout period, the sender will retransmit the data. By transmitting the data in numbered segments, only the missing segment needs to be transmitted rather than the entire message, saving on bandwidth.
- If the packets arrive in the wrong order, it does not matter as the sequence number can be used to rearrange the data into the correct order
- If a packet is received twice, the duplication can be detected and safely ignored [7] [8]

This is important for developers to take into account, as it enables them to lean on the guarantees of data ordering in TCP rather than building complicated mechanisms of ensuring data correctness themselves.

Flow Control

During a TCP connection, the receiving party can indicate to the sender how much data it can safely send without causing overflowing to the receive buffer. This allows the two parties to manage the flow of data meaning that data transmission is both efficient and robust. The receiver does this by telling the sender how many bytes it can safely receive beyond the current sequence number, known as the window principle. [7]

The Three Way Handshake

Communication using TCP begins with a TCP connection being established. TCP connections are established using a process known as the three-way handshake. In the first stage, the client makes initial contact with a server by sending a SYN packet, informing the server of the randomly-selected sequence number. In response, the server responds to the request with a SYN-ACK packet, and a sequence number. Finally, the client acknowledges the SYN-ACK packet with a single ACK. The connection is created, and both parties are able to transfer data. [7] [9]

The Two Generals' Problem is an established thought experiment in computer science that demonstrates the difficulty in achieving consensus over an unreliable communications medium. It states that it is impossible to ever truly guarantee that a message sent has been received and acknowledged at the other end. As the internet is inherently an unreliable communications channel, this poses a problem for computer networking in the real world. TCP attempts to solve this problem with the three-way handshake. While it is not possible to guarantee the receipt of a message after this process, it is reasonable to assume that after both the client and server send a single acknowledgment to each other, that the connection has been successfully established. Therefore, when developing applications where reliable receipt of messages is important, a developer should choose TCP to make use of this handshake.

Maximum Transmission Unit (MTU)

The Maximum Transmission Unit is defined as the largest size of a packet that can be sent across a link without requiring fragmentation.^[10] This is important as fragmentation results in additional network overhead and traffic, impacting network performance. MTU is represented as a number of bytes, with common values being 1500 (Ethernet) and 1492 (VDSL). TCP uses this value to determine the optimal packet size, improving performance through efficient use of network resources. MTU is often affected by the various protocols used within the OSI model. The use of different technologies at the various layers of OSI can change the resulting MTU value. It can be affected, for example, if Point-to-Point Protocol over Ethernet (PPPoE), or a Virtual Private Network tunnel (VPN) is used to encapsulate traffic. Different protocols have differing Protocol Data Unit sizes, meaning that the specific technologies in use will affect the final MTU size.

It is especially important to consider the impact of MTU when developing applications which are expected to have high network throughput such as distributed computing systems or network backup software. Excessive packet fragmentation in these environments could be detrimental to application performance, and investments in high speed transport mediums such as 10-gig or higher networking may not be fully realised. RFC1191 defines a mechanism for ICMP-enabled devices to automatically discover the MTU of a given network path.^[11] When developing such applications, developers should consider using the largest possible MTU to reduce the network overhead incurred. On some networks, developers can use Jumbo Frames, commonly defined as Ethernet packets with an MTU of larger than 2500 bytes. Often this is set to 9000 bytes, significantly improving the efficiency of network communication by making more space for user data while keeping the header size fixed.^[12]

Another reason that it is important to consider MTU size is for algorithms which depend on exact MTU information in order to function correctly. Such algorithms include Smart Queue Management (SQM) algorithms. These algorithms reduce network congestion and latency by scheduling packets based on network conditions, fair queueing, and QoS prioritisation. In order to work best, SQM algorithms need to know the MTU value of the upstream connection in order to estimate which packets are likely to fragment, consuming additional resources and introducing additional latency.^[13]

Synchronicity and Asynchronicity in Network Protocols

At a high level, synchronicity and asynchronicity refer to the way that systems communicate with one another. The terms describe how systems handle timing in their communication, and how they respond to each other. Synchronous communication typically has strict rules surrounding how and when each party communicates with each other. Asynchronous communication on the other hand, allows any party to send a message at any time, regardless of if the receiver is ready or even present, to receive it. For instance, TCP[4] could be viewed as a synchronous protocol, as it requires both parties to establish a session with each other before communication begins to occur (connection-oriented). This contrasts with UDP, which is connectionless.[5] UDP datagrams can be sent at any time, without first establishing a connection with the receiving party. This is important to application developers, as those seeking to develop a low-latency application which communicates quickly should opt for UDP, as there is no requirement to first establish a connection. For programs which prioritise reliability and correctness over speed, TCP is the preferred choice.

Another example of synchronous vs asynchronous communication is HTML5 Websockets.[14] [15] The websocket protocol allows for loose asynchronous communication between a web browser and a web server. Once a websocket connection is established, the client and server can send messages to each other at any time, bidirectionally. This obviates the need for the client to continuously poll the server in a synchronous fashion, making it more efficient and enabling real-time applications to deliver information instantly.

In order to make use of the asynchronicity offered by these protocols, it is important that developers write their software in a way that makes best use of these features. Threading is a concept that allows a program to handle tasks away from the main process, freeing it up to do multiple things at a time. This aligns well with asynchronous communication, as responsibility for handling connections can be offloaded to worker threads, freeing the main thread to do other things or indeed dispatch more connections. Programming languages have long offered the ability to manage threads, and newer languages are offering more modern ways to communicate asynchronously. For instance, JavaScript offers promises, which is a way of writing code in a way that is non-blocking. It also offers `async/await`, which is a shorthand wrapper around promises, making it easier to write asynchronous code.[16] Message queues allow actions to be carried out asynchronously, often splitting up work between a multitude of microservices or containers. Messages are placed into the queue, and can be picked up by a worker at any time in the future. This allows messages to be handled asynchronously by a team of worker nodes, increasing efficiency and scalability.[17]

Interoperability for Networked Applications

Interoperability is defined as the ability of systems or machines to work together with other systems or machines. Naturally, interoperability is important when thinking about connecting millions of computers from across the globe, or even two machines located

directly next to each other. Computer systems need to be able to speak a common language in order to understand each other. In order to participate in a shared language, it is important to follow a set of widely understood rules that govern how the language is spoken.

Over time, the global computing community settled on a set of standards that allowed for near universal compatibility of machines. Using a handful of defined mechanisms, typically known by acronyms, one can lookup the email address of somebody on the other side of the world and send them an email message. A person can wirelessly connect their laptop to a home network using WLAN. Their laptop can then obtain an address on the local network using DHCP. They can locate a website for looking up email addresses using DNS. They can then send an email to that address using SMTP, which ultimately relies on TCP.

Since all of these mechanisms are standardised, the process of sending an email to somebody is entirely seamless from the end user's point of view. They don't have to worry about whether the other person's laptop supports receiving simple messages over the internet, or even whether the device is a laptop at all! Interoperability allows us to forget about these sorts of intricacies, and allows us to focus on the task at hand.

Interoperability is best achieved through the adherence to standards, avoiding re-inventing the wheel.

Conclusion

In conclusion, it is important that programmers are aware of the range of networking protocols available and the benefits and caveats of each. Some protocols, such as UDP, are less concerned with reliability and more concerned with performance and low latency. Protocols such as TCP endeavour to provide guarantees that data is delivered reliably, at the cost of additional network and processing overhead. The 'Two Generals' problem reminds us that it is not possible to ever truly guarantee the receipt of a message, but in practice this is not usually a problem. The Maximum Transmission Unit impacts the maximum size of packets sent across a network, with implications for software that moves large amounts of data, such as backup software. Asynchronicity in network protocols allows communication to flow freely, with messages being sent and received at any time, allowing for more real-time and interactive experiences for the user. Interoperability and standardisation are allowing more applications to work together seamlessly, regardless of whether the user is on a laptop, PC, or mobile phone. It is crucial that we as programmers understand all of this, in order to make the end user's experience as good as it possibly can be.

References

- [1] J. Abbate, *Inventing the internet*. MIT Press, 2000.
- [2] V. Cerf and R. Kahn, "A protocol for packet network intercommunication," *IEEE Transactions on Communications*, vol. 22, no. 5, pp. 637–648, 1974, doi: [10.1109/TCOM.1974.1092259](https://doi.org/10.1109/TCOM.1974.1092259).
- [3] U. of S. C. Information Sciences Institute, "INTERNET PROTOCOL," RFC Editor; Internet Requests for Comments; RFC Editor, RFC 791, Sep. 1981. Available: <https://www.rfc-editor.org/rfc/rfc791>
- [4] U. of S. C. Information Sciences Institute, "TRANSMISSION CONTROL PROTOCOL," RFC Editor; Internet Requests for Comments; RFC Editor, RFC 793, Sep. 1981. Available: <https://www.rfc-editor.org/rfc/rfc793>
- [5] J. Postel, "User datagram protocol," RFC Editor; Internet Requests for Comments; RFC Editor, RFC 768, Aug. 1980. Available: <https://www.rfc-editor.org/rfc/rfc768>
- [6] G. F. L. Eggert NetApp, "UDP usage guidelines," RFC Editor; Internet Requests for Comments; RFC Editor, RFC 8085, Mar. 2017. Available: <https://www.rfc-editor.org/rfc/rfc8085>
- [7] L. Parziale *et al.*, *TCP/IP tutorial and technical overview*. in IBM redbooks. IBM Redbooks, 2006. Available: <https://books.google.co.uk/books?id=TgwWAgAAQBAJ>
- [8] GeeksForGeeks, "Wrap around concept and TCP sequence number." Accessed: May 08, 2025. [Online]. Available: <https://www.geeksforgeeks.org/wrap-around-concept-and-tcp-sequence-number/>
- [9] B. Leo, "TCP 3-way handshake (SYN, SYN-ACK,ACK)." Accessed: May 08, 2025. [Online]. Available: <https://www.guru99.com/tcp-3-way-handshake.html>
- [10] A. Custura, G. Fairhurst, and I. Learmonth, "Exploring usable path MTU in the internet," in *2018 network traffic measurement and analysis conference (TMA)*, 2018, pp. 1–8. doi: [10.23919/TMA.2018.8506538](https://doi.org/10.23919/TMA.2018.8506538).
- [11] S. D. J. Mogul DECWRL, "Path MTU discovery," RFC Editor; Internet Requests for Comments; RFC Editor, RFC 1191, Nov. 1990. Available: <https://datatracker.ietf.org/doc/html/rfc1191>
- [12] E. Alliance, "Ethernet jumbo frames." Accessed: May 08, 2025. [Online]. Available: <https://ethernetalliance.org/wp-content/uploads/2011/10/EA-Ethernet-Jumbo-Frames-v0-1.pdf>
- [13] OpenWRT, "SQM details." Accessed: May 08, 2025. [Online]. Available: https://openwrt.org/docs/guide-user/network/traffic-shaping/sqm-details#sqmlink_layer_adaptation_tab

- [14] G. L. Muller, "HTML5 WebSocket protocol and its application to distributed computing." 2014. Available: <https://arxiv.org/abs/1409.3367>
- [15] Inc. I. Fette Google, "The WebSocket protocol," RFC Editor; Internet Requests for Comments; RFC Editor, RFC 6455, Dec. 2011. Available: <https://www.rfc-editor.org/rfc/rfc6455.html>
- [16] Mozilla, "Promise." Accessed: May 20, 2025. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
- [17] N. Thallapally, "Enhancing distributed systems with message queues: Architecture, benefits, and best practices," *Journal of Electrical Systems*, 2025.

Final

GRADEMARK REPORT

FINAL GRADE

65/100

GENERAL COMMENTS

A reasonable report, though a little light on the detail in some areas.

A reasonable use of sources, though a few non-authoritative blog-type references included.

Still, a good effort overall.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9