

Report

by Silvia Settle

Submission date: 03-May-2023 12:00AM (UTC+0100)

Submission ID: 204421620

File name: 5780_Silvia_Settle_Report_499749_1672508783.docx (55.2K)

Word count: 2252

Character count: 13007

2023

Socket Programming and Concurrency

ACCA6028

SILVIA SETTLE 1708386

Contents

How the Transmission Control Protocol operates and endeavours to guarantee delivery of data. Refer to the three way handshake and so-called "Two Generals Problem".	2
The impact of the maximum transmission unit (MTU) size on the transfer of data. You should consider associated factors such as size of the protocol data unit (PDU)	3
Synchronicity and asynchronicity in network protocols. Consider the sort of applications you might use with each. Consider I/O dispatch models.....	4
Interoperability for networked application and how it is best achieved.	5

3000 words on the following topics:

Transmission Control Protocol – how it operates and endeavours to guarantee delivery of data. What is the three way handshake and the so-called “Two Generals Problem”.

The Transmission Control Protocol (TCP) is one of the two main Transport layer protocols of the Open Systems Interconnection Model (OSI). The main function of the Transport layer is the transfer of data between two nodes (devices) on a network and lies between the session layer (where a connection is established) and the network layer (which defines how packets are sent from one node to another) [2]. The purpose of the Transmission Control Protocol (TCP) is to provide reliable, in-order, byte-stream transmission service for networked applications [1]. The protocol endeavours to achieve this by first establishing a connection between the nodes in a process called the “three-way handshake”, in which the terms and conditions of the pending exchange are negotiated. The data packets transfer is then meticulously executed by labelling and sequencing them, and by closely monitoring the overall process to ensure no data loss occurs. At the start of a transmission, the first three exchanges between a client and a server are critical for establishing the details for how the rest of the data exchange will be completed (there is no data sent at this stage, just ‘meta-data’ called a header. The initial client request is labelled SYN - the client sends a request for data exchange detailing certain parameters/options they are able to support for the exchange such as the source and the destination port numbers, the initial sequence number (ISN) for labelling the sequence of packets, the window size (the size of packets the receiving party can handle determined by its cache-size), maximum segments size (MSS, the maximum length of the data field), the maximum transmission unit (MTU), etc. The initial response from the server (SYN-ACK) acknowledges the client’s request and advertises in turn the terms and conditions for the exchange the server is able to offer. The final confirmation of the conditions for the exchange come from the client (ACK) and are followed by the actual data transfer. Any irregularities in the exchange are monitored via mechanisms such as packets sequence numbers and checksums (the meaning behind the term ‘in-order’), until the end of the transmission is reached and a FIN flag is sent. At any point of time during the exchange, a request for a reset (RST) can also be sent, usually to indicate that something has gone wrong, meaning that if the whole process is to continue, then it must be restarted first, and re-negotiated [3][4]. Despite all the facilities for error checking that TCP offers to ensure that the data is delivered reliably, there is still a possibility that some of it may get lost. This challenge is demonstrated by the so called ‘two-generals problem’ [9]– a simplified example of a Byzantine failure on consensus in synchronous systems [8]. In this example, a communication problem is presented in which two generals from the same attacking army are trying to coordinate an attack, while being separated in two valleys by the opposing forces. The only mode of communication is by sending messengers, who could be captured, resulting in the message not reaching its destination. The paradox is that there is no straight-forward way of deciding how both sides can be 100% confident that the communication was successful and resulting in both sides attacking simultaneously. Similarly, TCP cannot guarantee 100% failure-free transmission, and these challenges must be acknowledged when considering the expectations of data transmission in fault-tolerant systems.

The impact of the maximum transmission unit (MTU) size on the transfer of data. You should consider associated factors such as size of the protocol data unit (PDU)

The Protocol Data Unit is the generic name for the data unit (message, packet, segment, frame) that is passed through the different layers of the OSI model [2]. Each layer will add and remove information from the PDU respectively.

As discussed earlier, when a device (node) on the network is requesting a connection through TCP, it advertises its window size to indicate the maximum data packet size it can handle; It also has a maximum segment size (MSS) to indicate the maximum size for the data part of a packet (minus the header) [1]. A network channel, on the other hand, has a MTU – Maximum Transmission Unit, determined by the bandwidth of data flow it can transfer.

One of the problems in network data transmission is that multiple messages (PDUs) will be travelling at the same time through the same network links, which may lead to collision, congestion and errors. Considering the potential variations in the connection channels between one node and another, a transmission over a network is ultimately a black-box process [5]. Instead of relying on data-link switched to repeat buffer feedback to indicate congestion, TCP has end-to-end flow-control mechanisms to ensure that the data transmission is fault-tolerant [10].

During transmission the source device/server on a network will be gradually increasing the packets window in a drive for efficiency (faster transfer), until the network becomes congested, and a packet is lost/dropped. While the User Datagram Protocol (UDP) protocol trades off error-handling for 'best-effort' but faster mode of transport, TCP endeavours to prevent data loss and guarantees the delivery of data with an in-built congestion control and congestion avoidance mechanisms [2]. TCP's active queue management prevents some of the negative consequences of queue overflows and the reliance on buffer overflow as the only measure of congestion [5]. What needs to be considered is how critical is the data in transit and how sensitive is an application to a data loss.

Synchronicity and asynchronicity in network protocols. Consider the sort of applications you might use with each. Consider I/O dispatch models.

What is synchronicity and asynchronicity?

Traditionally, almost all input/output requests within a single-core fetch-decode-execute systems are synchronous in nature – operations are performed sequentially, with the application standing idle while the input and output are ongoing, waiting for a confirmation if they were successful or not (von Neuman's architecture). Less common are the synchronous I/O requests-those that can be still in progress while the application thread continues, i.e. they will run in parallel [14]. The hardware limitations and the demand for more processing power forced the industry to seek different system architecture solutions such as multi-core processing, parallel computing, distributed, and interconnected/network-based systems [10]. Both of these models are still present in a higher-level systems architecture where devices and applications need to communicate and transfer data over a network, albeit in a different ratio [15]. The elements typically present in such systems are multiple processes, inter-process communication, disjoint address spaces and collective goal, therefore how all these come together is critical for their overall performance (and achieving this common goal) [8]. Network Protocols define the rules of how applications communicate with each other, and these are normally implemented through sockets as interfaces for exchanging information [16]. An application may require that the rules of communication to be synchronous, because the sequence of exchanged messages is important, or it is critical that they are all received. In such cases, protocols that provide more reliable exchange of data are suitable, such as SDCL at the data-link layer, TCP at the transport layer, or SOAP at the application layer. Among the characteristics of synchronous protocols are: use of external synchronous clocks, use of synchronous processes in a lock-step approach, use of bounded-delay channels, message order between the client and the server are the same, messages are only sent when the receiving party is ready to receive them [8]. Applications that would require such communication model focus on reliability and durability, for example banking systems, process-control industrial systems, medical systems. On the opposing end of the spectrum, we find applications that require quick-response and fast-performance, where speed is more important than reliability and checking the data integrity [8]. These systems are usually real-time, such as gaming or streaming services, data tends to flow in one direction at a time (half-duplex) and among the protocols used are UDP at the transport layer, IMAP or POP3 at the application layer.

Most of the time there are no purely synchronous versus purely asynchronous applications, but instead are the result of a careful consideration on challenges such as the degree of synchronisation, the network topology and how are the processes connected, what is the level of knowledge between processes, how are failures and time synchronisation handled, how is the system availability, mutual exclusion and leader election handled, as well as the system's replica management.

Interoperability for networked application and how it is best achieved.

Defining interoperability can prove to be a challenging task. At a high-level it can be interpreted as having two or more diverse systems communicating, sharing resources and working together to achieve a common goal [7]. Within the context of networked applications interoperability can be seen as the implementation of loosely coupled systems approach – the outsider has the perception of interacting with a singular system with a high degree of external interoperability, while internally the individual system elements, different sub-systems, or separate entities are kept private and isolated from one another and data, resources and communication are exchanged through interfaces [11]. At application development interoperability can be seen in the concepts of encapsulation and inheritance in object-oriented programming (OOP) where entities private attributes are only visible or accessible to the rest of the overall system through public methods; and the Liskov principle of being able to replace any sub-class with its base class without any change of behaviour [13].

Interoperability can be also defined within different contexts and levels of abstraction – are we purely concerned with technical interoperability (at infrastructure and protocol level), or syntactic interoperability (data formats encoding), or semantic interoperability (understanding the information that is exchanged), or organisational interoperability (the ability to effectively communicate and share resources to achieve a common goal at organisational level) [12].

The net-centric approach interprets interoperability as “software applications and internet-based information services that are distributed over a globally-spread environment comprised of interoperable computing and communication components that aim to achieve optimal use of resources” [p.2, 11]. However, if some of the challenges that face interoperability are that system parameters change frequently and are difficult to predict, that systems must remain operational at multiple organisational levels, and that exchanges are complex and often use proprietary, custom interfaces, where do we draw the line on system boundaries and how can we ensure consistent and reliable behaviour? Measuring and achieving interoperability can prove therefore equally challenging. Cloud computing can be seen as a practical implementation of interoperability – services are scalable, resilient, accessible, and on-demand. It appears logical therefore to measure the level of system interoperability through the following parameters: how easy is to add new elements; can the order of the elements be changed; how easy is to change the structure/the names of the elements; can elements be omitted? [11]. It would also appear that the exact challenges described above have driven the current interoperability solutions towards the use of application programming interfaces (APIs)? After all, they are described as self-contained, modular packages that provide useful and independent functionality that when combined together allow software application to be constructed as interoperable, standard-based services with well-defined interfaces and layers, and making better use of operational and technical standards [11]. That is, until even better solution is found in this apparent continuum of pendulum-like sway of changes from centralisation to de-centralisation and back.

Bibliography

There are no sources in the current document.

- [1] Eddy, W. ed RFC 9293: Transmission Control Protocol (TCP), 2022, online www.ietf.org/rfc/rfc9293.html [accessed on 11/04/2023]
- [2] B. A. Hallberg, Networking: a beginner's guide, Sixth edition. New York, New York State: McGraw-Hill Education, 2014.
- [3] Chris Greer <https://packetpioneer.com/>
- [4] ICT Data Communications and Network Technologies, Huawei Technologies Co. Ltd, Posts & Telcom Press, 2023
- [5] Ramakrishnan K. et al. The Addition of Explicit Congestion Notification (ECN) to IP. Network Working Group, 2001
- [6] Manzanares-Lopez, et al. Empirical and Analytical Study of a Multicast Synchronous Transport Protocol for Intra-Campus Replication Services, IEEE International Conference on Communication. Communications Society, pp.1461-1466. 2004
- [7] T. R. Gruber and et al., A translation approach to portable ontology specifications, Knowledge acquisition, 1993
- [8] Ghosh, Sukumar. Distributed Systems. An algorithmic Approach, Chapman & Hall/CRC Computer and Information Science Series, 2007
- [9] Jim Gray Summary Home Page.
<http://research.microsoft.com/~Gray/JimGrayHomePageSummary.htm>
- [10] Rauber, T, et al. Parallel Programming for multicore and Cluster Systems, Second Edition, Springer, 2013
- [11] Ghosh, Supriya. *Net Centricity and Technological Interoperability in Organizations: Perspectives and Strategies*. Hershey, PA: IGI Global, 2010.
<https://doi-org.ezproxy.uwtsd.ac.uk/10.4018/978-1-60566-854-3>
- [12] Vermesan, O. (Ed.). *Advancing IoT Platforms Interoperability*. First edition, River Publishers, 2018, <https://doi.org/10.1201/9781003337126>
- [13] ...
- [14] Yosifovich, P, et. al. Windows Internals, Part 1: System architecture, processes, threads, memory management, and more, 7th edition, Pearson, 2017
- [15] A. Ros and A. Ros, *Parallel and distributed computing*. Rijeka, Croatia: IntechOpen, 2010.
- [16] Donahoo, M., K. Calvert, *TCP/IP Sockets in C. Practical guide for Programmers*, Morgan Kaufmann Publishers, 2001.

Report

GRADEMARK REPORT

FINAL GRADE

62/100

GENERAL COMMENTS

Instructor

A good effort, with a well-written and presented report, with a good use of sources.

A good effort with the implementation, a rock paper scissors game provided - I suspect you could've taken this a bit further from a technical perspective, though admittedly there's only so much you can do with rock paper scissors!

Part 1 (report): 60

Part 2 (implementation): 64

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7