

final

by Tirion Wall

Submission date: 15-May-2025 12:05PM (UTC+0100)

Submission ID: 259353979

File name: 27859_Tirion_Wall_final_664976_1766658683.docx (378.24K)

Word count: 3545

Character count: 21220

Tirion Wall
Student number: 2023509
Socket Programming and Concurrency
Report (ACCA6028)
May 2025

Contents

Introduction	3
How the Transmission Control Protocol operates and endeavours to guarantee delivery of data	3
Synchronicity and asynchronicity in network protocols.....	5
The meaning of interoperability for networked applications.....	6
The impact of the maximum transmission unit (MTU) size on the transfer of data	8
Conclusion	10
References	11

Introduction

This report explores how the Transmission Control Protocol (TCP) works & how it strives to guarantee delivery of data. It also looks at the impact of maximum transmission unit size on the transfer of data, synchronicity and asynchronicity in network protocols, and the meaning of interoperability for networked applications. It considers how each of these are relevant to programmers when developing networked applications.

How the Transmission Control Protocol operates and endeavours to guarantee delivery of data

Transmission Control Protocol

Transmission Control Protocol (TCP) and Internet Protocol (IP) are protocols that together allow devices to communicate and exchange data over networks. While IP is responsible for getting each packet to its target, TCP ensures that the data arrives in the correct order, complete and error free [1]. TCP breaks large amounts of data into multiple packets. Each packet includes a header with details such as its destination and the order in which it should be delivered. TCP's guarantee of delivery is important for developers creating networked applications that need reliable and sequential data transmission, such as messaging services.

TCP also uses acknowledgements to check that packets have been received. If a packet isn't confirmed, it gets automatically resent. Each packet has a sequence number so the receiver can reassemble packets in the correct order upon arrival [2], even if they arrive in an incorrect order. TCP also uses flow control mechanisms so that the receiver isn't overwhelmed with too much data at once, which helps avoid potential data loss [2]. Congestion control is another mechanism used by TCP to manage how much information is sent, dynamically adjusting the volume of data to help prevent the network from becoming overwhelmed [3].

User Datagram Protocol (UDP) is a connectionless protocol that works faster than TCP. However, in UDP, the order of the messages is not guaranteed and there is no message acknowledgment [4]. UDP is therefore faster but less reliable. TCP is the best option when reliability is key (for example, messaging services), whereas UDP is better suited for real-time applications such as online gaming [5].

To ensure two devices (for example, a laptop and a server) are ready to communicate before exchanging any data, TCP uses what is known as the three-way handshake, which is a three step process that establishes a connection between the two devices on a network. The connection is full duplex, and both sides synchronize (SYN) and acknowledge (ACK) each other" [6]. A full-duplex system means both devices can simultaneously send and receive data.

First, a SYN packet is sent from the client with a random sequence number. The server then replies with a SYN-ACK packet, which contains a random sequence and ACK number. Finally, the client sends an ACK number to confirm the connection [7].

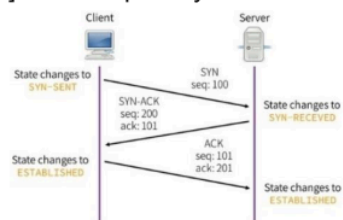


Figure 1. Three-way handshake steps [7]

Three-way handshake

The three-way handshake works in the following three steps:

1. To establish a connection, the client sends the server a SYN (synchronise packet)
2. A SYN-ACK packet is sent from the server. The ACK confirms it received the client's SYN packet, and SYN shows the server's starting sequence number
3. The connection is made once the client confirms with an ACK packet

The three-way handshake is particularly relevant to programmers when developing networked applications as it helps ensure that network connections are safe and prepared for communication. Additionally, each step of the process includes an acknowledgement, which helps detect any issues in the connection and helps reduce security risks. Sequence numbers also help keep the data in the correct order while data is being exchanged.

Two Generals' Problem

The challenge being posed is to ensure trusted communication over a network that cannot be assumed to be reliable. This is illustrated in the thought experiment known as the Two Generals' Problem. In order to achieve a successful attack, the two generals, in separate locations, need to co-ordinate the time of attack. The only way for them to communicate is via messengers, meaning the communications could be intercepted or lost, in turn leading to uncertainty between the generals of what communications have been received or lost. If failure can occur at any point during communication, it becomes impossible to guarantee delivery. Therefore, programmers need to take this into account when accurate delivery of messages is the primary goal of the networked application being developed. TCP provides acknowledgements and retransmissions to work around this issue. However, although TCP is very reliable, "it doesn't solve the Two General's Problem because any remote communication is not 100% reliable" [8].

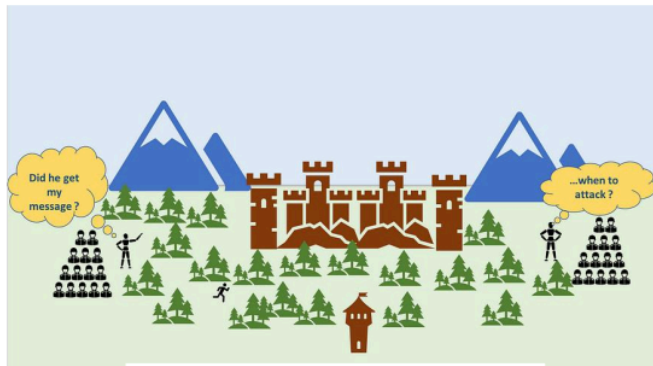


Figure 2. The Two Generals' Problem [9]

TCP ensures data gets sent in the correct order over a network, and it can recover from delays and lost packets. However, as the Two Generals' Problem highlights, network communication is never 100% reliable. Programmers also need to decide whether UDP is more effective for the type of application being built. For example, UDP is better for real-time applications like live streaming as it is faster than TCP, whereas, TCP is more appropriate for things like file transfers and sending emails, where it is vital they are sent. TCP can be slower because the three-way handshake takes time to establish a connection and can timeout if there's no response.

Synchronicity and asynchronicity in network protocols

When developers are designing networked applications, it's important to understand how data transmission is handled. In synchronous data exchange, the sender waits for a response before continuing, whereas the sender can continue without waiting in asynchronous data exchange [10]. Asynchronous communication can therefore handle more at once and the latency is reduced, which improves the user experience, while synchronous communication is easier to code but it can be slower in larger applications. This can have a big impact on an application's speed.

Synchronous communication is a blocking architecture, where the first job must finish "before moving on to the next"[11], whereas asynchronous communication is non-blocking, meaning the system can still run while tasks are in progress [11]. According to Microsoft, "Synchronous: The client sends a request and waits for the response. This is the way HTTP is behaving" and "Asynchronous: The client does not wait for a response and just sends the request to a message broker. AMQP is an asynchronous protocol" [12]. HTTP is used when a user and website exchange data, and AMQP (Advanced Message Queuing Protocol) is a protocol that allows applications to communicate without waiting for a reply (for example, sending an email).

Synchronous I/O and asynchronous I/O are two types of synchronisation. In synchronous I/O, the I/O operation is initiated once a thread starts the task, and then waits until it's complete, blocking other processes in the meantime. In asynchronous I/O, the thread can continue processing while the operation is still running [13]. A thread divides a program's operations into smaller, independent tasks, allowing the system to run them simultaneously.

This is managed by an I/O dispatch model which handles system input and output operations. One example is the thread pool model, which keeps a bunch of threads ready to go whenever tasks come in, meaning the system isn't constantly setting up and stopping threads for each task, which can slow things down. This is particularly useful when lots of I/O operations are running at the same time. This is a type of asynchronous model because it can handle multiple I/O tasks at once without stopping everything from running. As previously mentioned, synchronous models handle one task at a time. An example of a synchronous model is thread-per-request, where each thread handles a task at a time from start to finish.

Although asynchronous can be harder to write, `async/await` syntax is now available in most programming languages [14]. Asynchronous uses less resources as it does not require a new thread for each task, whereas synchronous uses more threads, which take up more memory under a heavy load.

Many applications use both asynchronous and synchronous communication. Asynchronous is better for things like emails, direct messaging, or pre-recorded videos as they don't need an immediate response. Synchronous is more suitable for APIs and database queries, where data needs to be returned straight away. Developers are encouraged to use synchronous communication for tasks that immediately affect the user experience (e.g. balance updates), and asynchronous communication for tasks that can be completed later without disruption (e.g. calculating fees). In large applications, it's often better to use both, so each model can focus on what it's best at, preventing system overload [15].

Deciding between synchronous and asynchronous programming depends on the application. Synchronous programming is more appropriate for smaller systems as it's easier to write and debug, and it's better at managing simple tasks, while asynchronous is better at handling long-running tasks, and provides more scalability and has reduced thread overhead.

The meaning of interoperability for networked applications

Interoperability in networked applications is essential for enabling systems to exchange data and communicate with other systems and hardware. H. Panetto and J. Cecil explain that interoperability is more cost effective and quicker to implement

in “decentralised, flexible and networked system environments” [16]. Most services that we use on a regular basis, such as healthcare, banking, and online shopping, rely on different systems collaborating with each other. For example, a lab may use one system to process test results, and another to share the results with healthcare providers. As NHS England highlights, “lack of access to an up-to-date medical history can affect the way patients are treated. The impact on treatment, particularly in an emergency situation, can be significant [17]”.

In 2017, Russia launched the NotPetya cyberattack on Ukraine, disguised as a tax software update. It had a massive impact on Ukraine’s infrastructure - ATMs, gas stations, emails, payroll systems, and even Chernobyl’s radiation monitors were shut down. Ukraine not being fully automated helped prevent even worse damage. Highly interconnected nations like the U.S, where systems are web-enabled, are much more vulnerable [18]. While interoperability makes our lives easier, this attack highlights the importance of building secure interconnected systems.

Interoperability in decentralised systems means separate systems, that may vary in programming languages, operating systems, and hardware, can work together within the same environment. For this to work, they must be built around the same set of standards, such as JSON, REST, HTTP, MQTT, and XML. These standards usually have shared data formats and agreed ways to transfer data. They also normally include an application programming interface (API), which connects the systems [19], using formats such as XML or JSON and communication protocols such as HTTP and TCP/IP. JSON and XML allow data to be sent between systems in a clear format, and HTTP and TCP/IP ensure that the data is reliably sent.

Middleware makes it easier for separate databases, systems, and applications to share data – it makes sure that the data is passed in way all systems can understand and use. Additionally, it can encrypt data to make it secure, balance loads, and transform and format data. With an API standard in place, systems only need to create one connection to link up with different endpoints, regardless of whether they’re other networks, or third-party apps.

It’s important for programmers to prioritise interoperability when designing networked applications. According to the IEEE Computer Society, organisations used eight software-as-a-service solutions (SaaS) in 2015, they now use 110 [20]. It can be challenging when more services are added, for example, “interoperability challenges at the application level might arise due to the different ways the application developers might choose to implement the program functionality, including different use of the underlying middleware. [21]” Developers need to test how their systems work with others, comply to shared standards, and keep their documentation clear and up to date. Interoperability makes development easier because new systems can be added to existing ones, without rebuilding everything from scratch. It’s important to consider interoperability in the design process to avoid compatibility issues later, and to save time and money.

Semantic interoperability ensures that systems understand the data they're exchanging, S. Albouq et al. explain that it's "a way to understand the meaning of the messages by gestures or tokens, and therefore provides a higher level of cooperation during the process of exchanging data"[22]. Schema.org and Resource Description Framework ensure that systems can share the same vocabulary, so that the data is understood in the same way [22].

Interoperability is essential in modern systems. It allows us to make better decisions, stay connected, and keeps data moving in the background without needing to get involved. As we rely more on online banking, healthcare, shopping, and other services, making sure systems can exchange data is vital.

The impact of the maximum transmission unit (MTU) size on the transfer of data

Maximum transmission unit (MTU) is the size of the largest data unit (for example, a frame or a packet) that can be transmitted in a single transaction. Due to backward compatibility, the MTU size has remained unchanged at 1500 bytes for many years [23].

Protocol Data Units (PDUs) are units of data that share information from one system to another. They are part of a layer of the OSI (Open Systems Interconnection) model and include the data required for that layer to properly work. Every layer adds its own header to the PDU, containing data from the layer above. This means each layer can handle specific tasks for data transmission. At the data link layer, the PDU is called a frame, at the network layer it's a packet, and the transport layer is known as a segment [24].

As the data moves down the OSI layers, each one adds its own header and potentially more data. If the final size goes over the MTU limit, the unit is broken into smaller units, a process known as IP fragmentation. This can happen with small amounts of data, if each layer is adding its own header.

When a host sends an IP packet over the network, it cannot exceed the largest size that the local network allows. The size of this is determined by its IP MTUs, which are often the same [25].

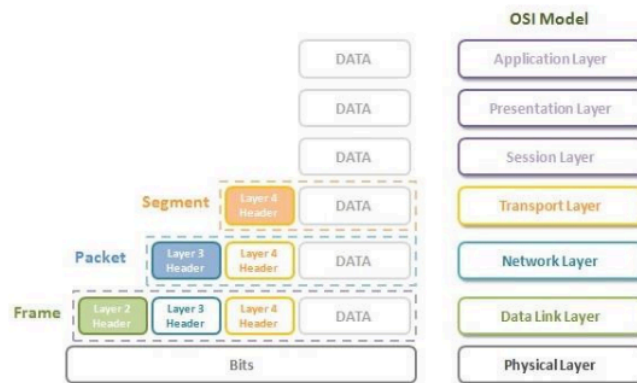


Figure. 3. OSI model [26]

Fragmentation comes with drawbacks: it can reduce performance speed, make packet loss more likely, and it can cause re-ordering of the data when it arrives. Additionally, it has its vulnerabilities, hackers can perform DoS attacks by overloading servers with packets that appear fragmented. If the server keeps waiting for packets that never arrive, it will eventually crash.

IP fragmentation happens at Layer 3 of the OSI model, which dictates which path the data will take. Each fragment has its own header that contains a unique ID to link the fragments to the original packet, flags to show the order of the fragments, and finally an offset to show where each fragment is positioned [27].

TCP is more reliable than UDP when it comes to packet loss as it makes sure each packet is delivered, and if one isn't, it's automatically resent. As mentioned previously, the order of messages is not guaranteed in UDP and it will not resend lost packets.

Y-J. Lin et al.'s study showed that increasing their UDP packet size from 1472 to 1473 bytes caused a noticeable drop in throughput, due to IP fragmentation [28].

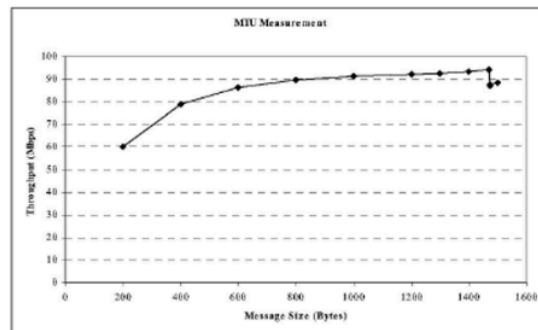


Figure. 3. Drop in throughput by increasing the UDP packet size by 1 byte

They also found that increasing the message size made errors more likely – when it went from 200 to 1000 bytes, the packet error rate jumped from 17% to 83% [28]. This demonstrates how using large packet sizes can increase the chances of lost data, and slow performance down. This is especially relevant today, as many of us work remotely and rely on secure VPNs. Because VPNs add encryption, they can reduce the available MTU, meaning fragmentation is more likely. It's therefore important for developers, when designing networked applications, to divide large files into smaller, independent units to help avoid fragmentation. C. A. Kent and J. C. Mogul also explain that “loss of a single fragment requires the higher level protocol to retransmit all of the data in the original datagram, even if most of the fragments were received correctly” [29]. Developers should consider using Path MTU Discovery when designing networked applications as it helps identify MTU on the network path and avoid fragmentation.

However, larger frame sizes are sometimes more beneficial. For example, researchers at Murdoch University found that “using 9000 byte frames, 99% of the time is spent transmitting payload data, with only 1% used for headers. With regular 1500 byte frames, 94.3% of the time is used for transmitting data [30]”.

Conclusion

In conclusion, it's important to understand how multiple network components work together. You need to combine all of the different areas, such as TCP, UDP, MTU, synchronous and asynchronous communication together to make applications work efficiently. Developers can build more efficient networked applications when they understand how these work together.

References

- [1] Imperva, "Transmission control protocol (TCP)", *Imperva*, [Online]. Available: <https://www.imperva.com/learn/ddos/tcp-transmission-control-protocol/>. [Accessed: Apr. 13, 2025].
- [2] P. Waite, "Understanding Transmission Control Protocol: The Backbone of the Internet", *Wray Castle*, Aug. 20, 2024. [Online]. Available: <https://wraycastle.com/blogs/knowledge-base/what-is-transmission-control-protocol?srsId=AfmBOoohzbJBcxwQbaxwA8DUGU-KdnLPwYUYkBs2cx6ZYEp-hSxjCms>. [Accessed: Apr. 13, 2025].
- [3] J. Lake, "Understanding TCP Congestion Control," *Orhan Ergun*, May. 6, 2024. [Online]. Available: <https://orhanergun.net/tcp-congestion>. [Accessed: Apr. 13, 2025].
- [4] F. T. AL-Dhief, et al, "Performance Comparison between TCP and UDP Protocols in Different Simulation Scenarios," *International Journal of Engineering & Technology*, vol. 7, no.4.36, pp. 172-176, 2018. [Online]. Available: https://www.researchgate.net/publication/329944111_Performance_comparison_between_TCP_and_udp_protocols_in_different_simulation_scenarios.
- [5] B. Gorman, "TCP vs UDP: Difference between the protocols," *Avast*, Feb. 23, 2023. [Online]. Available: <https://www.avast.com/c-tcp-vs-udp-difference>. [Accessed: Apr. 13, 2025].
- [6] W. Goralski, *The Illustrated Network*, Morgan Kaufmann, 2009. Available: <https://www.sciencedirect.com/topics/computer-science/three-way-handshake>. [Accessed: Apr. 14, 2025].
- [7] F.-H. Hsu, Y.-L. Hwang, C.-Y. Tsai, W.-Tai. Cai, C.-H. Lee, K. Chang, "TRAP: A Three-Way Handshake Server for TCP Connection Establishment," *Appl. Sci*, vol. 6, no. 11, p. 358, Nov. 2026. [Online]. Available: <https://doi.org/10.3390/app6110358>.
- [8] A. Gelenler, Two Generals' Problem and Idempotency, *Medium*, Sep. 26, 2024. [Online]. Available: <https://medium.com/@ali.gelenler/two-generals-problem-and-idempotency-4d28f4b07694>. [Accessed: Apr. 19, 2025].
- [9] A. Kapoor, "Kafka — What is 2 Generals' Problem? And can it affect my Kafka Producer?", *Medium*, Sep. 2, 2023. [Online]. Available: <https://medium.com/codex/kafka-what-is-2-generals-problem-and-can-it-affect-my-kafka-producer-2a621d4e1fbd>. [Accessed: Apr. 19, 2025].
- [10] "Synchronous vs. Asynchronous Communication – System Design", *Geeks for Geeks*, Sep. 23, 2024. [Online]. Available: <https://www.geeksforgeeks.org/synchronous-vs-asynchronous-communication-system-design/>. [Accessed: Apr. 19, 2025].

- [11] D. Bevans, "Asynchronous vs. Synchronous Programming: Key Similarities and Differences", *Mendix*, Oct. 14, 2024. [Online]. Available: <https://www.mendix.com/blog/asynchronous-vs-synchronous-programming/>. [Accessed: Apr. 19, 2025].
- [12] A. Zaeimi, "Synchronous vs Asynchronous communication in microservices integration", *Medium*, Apr. 1, 2020. [Online]. Available: <https://medium.com/@mmz.zaeimi/synchronous-vs-asynchronous-communication-in-microservices-integration-f4dd36478fd2>. [Accessed: Apr. 19, 2025].
- [13] Microsoft, "Synchronous and Asynchronous I/O", *Microsoft Learn*, Mar. 20, 2021. [Online]. Available: <https://learn.microsoft.com/en-us/windows/win32/fileio/synchronous-and-asynchronous-i-o>. [Accessed: Apr. 19, 2025].
- [14] E. Retunsky, "Distributed Systems and Asynchronous I/O", *Medium*, Nov. 18, 2020. [Online]. Available: <https://xnuter.medium.com/distributed-systems-and-asynchronous-i-o-ef0f27655ce5>. [Accessed: Apr. 19, 2025].
- [15] L. Comanescu, "Synchronicity vs. Asynchronicity: The Art of Building FinTech That Works", *LinkedIn*, Feb. 5, 2025. [Online]. Available: <https://www.linkedin.com/pulse/synchronicity-vs-asynchronicity-art-building-fintech-works-comanescu-fbeff>. [Accessed: Apr. 20, 2025].
- [16] H. Panetto and J. Cecil, "Information systems for enterprise integration, interoperability and networking: theory and applications," *Enterprise Information Systems*, vol. 7, no. 1, pp. 1-6, 2013, doi: <https://doi.org/10.1080/17517575.2012.684802>
- [17] NHS England, *Interoperability*, Version 1.2, Mar. 13, 2025. [Online]. Available: <https://www.england.nhs.uk/long-read/interoperability/>. [Accessed: Apr. 20, 2025].
- [18] N. Perlroth, *This Is How They Tell Me The World Ends*, London, UK: Bloomsbury Publishing, 2021.
- [19] G. Lindemulder and M. Kosinski, "What is interoperability?", *IBM*, Jan. 22, 2025. [Online]. Available: <https://www.ibm.com/think/topics/interoperability>. [Accessed: Apr. 19, 2025].
- [20] S. Flynn, "Allowing Solutions to Speak to One Another – The Importance of Interoperability", *IEEE Computer Society*, Feb. 09, 2022. [Online]. Available: <https://www.computer.org/publications/tech-news/trends/importance-of-interoperability>. [Accessed: Apr. 21, 2025].
- [21] G. Blair, M. Paolucci, P. Grace, and N. Georgantas, *Interoperability in Complex Distributed Systems*, in *Connectors for Eternal Networked Software Systems*, M. Bernardo and V. Issarny, Eds. Berlin, Germany: Springer, 2011. [Online]. Available:

<https://inria.hal.science/inria-00629057v1/file/InteroperabilityComplexDistributedSystems.pdf>.

[22] S. S. Albouq, A. A. A. Sen, N. Almashfi, M. Yamin, A. Alshanqiti, N. M. Bahboub, "A Survey of Interoperability Challenges and Solutions for Dealing With Them in IoT Environment", *IEEE Access*, vol. 10, Jan. 2022, doi: [10.1109/ACCESS.2022.3162219](https://doi.org/10.1109/ACCESS.2022.3162219)

[23] P. Reviriego, A. Sánchez-Macián, J. A. Maestro and C. J. Bleakley, "Increasing the MTU size for Energy Efficiency in Ethernet," *IET Irish Signals and Systems Conference (ISSC 2010)*, Cork, 2010, pp.124-128, doi: 10.1049/cp.2010.0499

[24] *Ascent Optics*, "Understanding Protocol Data Units (PDUs) in Networking and Power Distribution Units", Jan. 6, 2025. [Online]. Available: <https://ascentoptics.com/blog/pdu-networking/>. [Accessed: Apr. 21, 2025].

[25] S. Iveson, "IP fragmentation in detail," *Packet Pushers*, Nov. 26, 2019. [Online] Available: <https://packetpushers.net/blog/ip-fragmentation-in-detail/> [Accessed: May. 06, 2025].

[26] "OSI Reference Model," *IPCisco*. [Online]. Available: <https://ipcisco.com/lesson/osi-referance-model/> [Accessed: May. 01, 2025].

[27] A. Stephenson, "What is an IP Fragmentation DDoS Attack?," *Corero*, Feb. 21, 2025. [Online]. Available: <https://www.corero.com/what-is-ip-fragmentation-ddos-attack/> [Accessed: May. 02, 2025].

[28] Y.-J. Lin and H. Latchman, "On the Effects of Maximum Transmission Unit in Power Line Communication Networks," *2007 IEEE International Symposium on Power Line Communications and Its Applications*, Pisa, Italy, 2007, pp. 511-516, doi: 10.1109/ISPLC.2007.371177

[29] C. A. Kent and J. C. Mogul, "Fragmentation considered harmful," *ACM SIGCOMM Computer Communication Review*, vol. 25, no. 1, Dec. 1987, doi: [10.1145/55483.55524](https://doi.org/10.1145/55483.55524)

[30] D. Murray, T. Koziniec, K. Lee, and M. Dixon, "Large MTUs and internet performance," *2012 IEEE 13th International Conference on High Performance Switching and Routing*, Belgrade, Serbia, 2012, pp. 82-87, doi: 10.1109/HPSR.2012.6260832

FINAL GRADE

GENERAL COMMENTS

70/100

A good effort overall. Do consider your use of sources, however - medium, especially, is essentially a blogging site and not authoritative. Try to make use of primarily peer reviewed material.

A well written report.

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