

[Year]

Socket Programming

256602 (JAKE MASON)

Transmission Control Protocol (TCP)

How TCP operates

TCP delivers streams of bytes between applications in a reliable manner, error checking the data before delivery. Emails, remote administration and file transfer depend on TCP. A connection should be established between the client and the server before data is sent, the server should then be open for connection requests. TCP breaks large data packets into smaller bits, ensuring its integrity remains throughout the transfer process. There are four layers to TCP: datalink, internet, transport and application [1].

1. The datalink layer administrates the act of sending and receiving data, responsible for transmission between applications and devices on a network [1].
2. The internet layer handles the sending of packets from a network, and its movement across a network, to guarantee it reaches its intended destination [1].
3. The transport layer provides a data connection between the sender and the receiver. This layer is where data is divided into packets and put into a sequence [1].
4. Application layer refers to applications that require TCP to allow for communication between them – programs here include email and messaging platforms [1].

Commented [AG1]: Using the simple OSI model is acceptable.

Delivery of data

TCP uses three schemes to deliver data between two different internet hosts: addressing, routing and multiplexing. [2] [1]

1. Addressing entails IP addresses being used to identify hosts on a network, to make sure data is sent to the right host. [2] [1]
2. Routing uses gateways to send data to the correct network [2]. A route defines the segments of a path in which packets are sent through. Gateways operate on the network interface, receiving IP datagrams from other hosts and routing them from one IP to another [3] [4].
3. Multiplexing is the process in which multiple processes needing to send and receive datagrams are handled on the same IP, essentially the information is “funnelled down” to the transport layer, before being sent to their different intended destinations [5] [6] [7] [8]. Port and protocol numbers send information to the correct module within the host [2] [1].

Three-way handshake

The three-way handshake in the context of TCP is a ‘handshake’ between the client and the server when beginning or concluding a TCP connection [9]. The three-way handshake assists in initiating, negotiating and separating TCP socket connections simultaneously, allowing for multiple socket connections to be transferred in both directions simultaneously. there are three steps requiring the client and server to synchronize and acknowledge packets before the process of data communication can begin [10].

1. Connection with server is established. Segment is sent with SYN; server is informed of client and communication begins [10].
2. The server responds to the client request with a SYN-ACK signal set, ACK signifies the received segment’s response, SYN represents the sequence numbers that should be used to start with the segments [10].

3. Client acknowledges server's response; a stable connection is created, and the data transfer process starts [10].

Message Type	Meaning
SYN	Establish connections and synchronize sequence numbers between different devices.
ACK	Confirms that SYN has been received.
SYN-ACK	Server received SYN from client (ACK is then sent back to server).
FIN	Ends connection

[10]

Commented [AG2]: This is all correct but is purely descriptive and over-uses lists

"Two Generals' Problem"

The "Two Generals' Problem" claims there is no way of knowing if the second party has received the first party's messages, if they are communicating over an unreliable network that is [12]. The endpoint in a TCP connection makes assumptions regarding the other party without directly knowing the state of said party, an assumption is always initially made that the connection will work, but the opinion will then change after (for example) a timeout – TCP cannot guarantee state consistency between endpoints [11].

Commented [AG3]: Agreed

Relevance to programmers

Knowing TCP is relevant to programmers developing networked applications as it defines the relationship between client and server. Knowing the three schemes utilised in the delivery of data can help programmers troubleshoot connection issues and find out what is causing data to not be delivered correctly, if at all. The three-way handshake is relevant for similar reasons, displaying how connections are established and how TCP connections are negotiated, having a more enriched understanding can make the development of programs which communicate between a client and server more effective in producing more functional programs. Acknowledgement of the 'Two Generals' Problem' is important to understanding possible communication consistency issues such as timing out, which are common; TCP cannot guarantee consistency between two different endpoints, which is often assumed to be the case with most systems of communication on a surface level [12], so knowing that this issue exists can alleviate frustration and help understand what an issue with consistency may be.

Commented [AG4]: Fine, brief but there is demonstration of understanding here.

Impact of Maximum Transmission Unit size (MTU)

The MTU is the largest Protocol Data Unit (PDU) in a sole network layer interaction. Larger MTU sizes allows less packets equating to a larger size to reach the same network throughput – larger packets reduce the processing power needed by the OS to allow data to be sent, a larger MTU allows higher efficiency as each network packet can carry more data as protocol overheads will stay fixed. Smaller MTU values however can reduce delays in the network, this is because large packets "occupy a small link" for longer, increasing delay variation [13]. Large packets can cause issues when communication errors are present, as corruption of a "single bit" in a packet will need the whole packet to be transmitted from scratch, which can take a while, larger packets have a higher chance of becoming corrupted [14]. Despite higher chances of corruption, larger packets are said to have a positive effect on "end-to-end TCP performance" [15].

Commented [AG5]: True, but this is not explained.

Position in OSI model

The transport layer is responsible for data transmission using transmission protocols, data being broken into segments before transmission occurs here, flow control is handled in the transport layer also [16]. The functions which are affected by MTU take such as transmission speed, retransmissions and packet segmentation occur in the transport layer [13] [16]. Its position in the OSI model is important to consider when it comes to assessing the impact of the MTU because it details which features in the OSI model are unaffected by it – those being the functions which occur in lower levels than wherever MTU's impact comes into consideration, in this case MTU is in the transport layer – thus MTU has little to no impact on the network, data link or physical layer; the layer immediately after transport is session, which is responsible for opening sessions between devices, a session between devices is pointless to create if the data being segmented is for example in some way corrupted [17].

IP Fragmentation

IP fragmentation is the breaking down of packets into smaller bits called 'fragments', this is done so these packets can pass through a link with a smaller MTU than the original packet size. The fragments are made whole again by the host who receives the data on the other end [18]. IP fragmentation allows higher layer protocols and IPs to function throughout a diverse set of network paths without a path discovery protocol being in place – IP fragmentation also allows for transport layer protocols to "downsize overheads" and ignore network architecture [27].

The architecture and mechanism in which fragmentation occurs is done differently on IPv4 and IPv6 addresses [22] [20] [21] – in IPv4 hosts try to reassemble IP packets with a reassembled size of up to 576 bytes, while in IPv6 this size is 1500 – attempts to reassemble larger sizes in IPv4 are attempted, but they are permitted to "discard" larger packets, applications are advised to avoid sending packets larger than 576 unless the remote host is capable of reassembly, and this fact is known. Applications which use IPv6 to fragment to overcome MTU limits will fragment the packet "at the point of origin". Packets with a "reassembled size larger than 1500" are discarded [19].

Commented [AG6]: No!

IP fragmentation is considered "fragile" and undesirable due to its impact on security [23]. Fragmentation can lead to excessive retransmissions as fragments see packet loss and protocols like TCP will need to transmit every fragment in an attempt to recover the loss of even a single fragment [24]. To avoid fragmentation the Path MTU Discovery algorithm is run to find the path MTU between two hosts [25]. Another way IP fragmentation is avoided is for the host sending the packet to send an IP Packet which is the same size as the MTU of source-destination pair's first hop [26].

Commented [AG7]: No discussion of PDU

Relevance to programmers

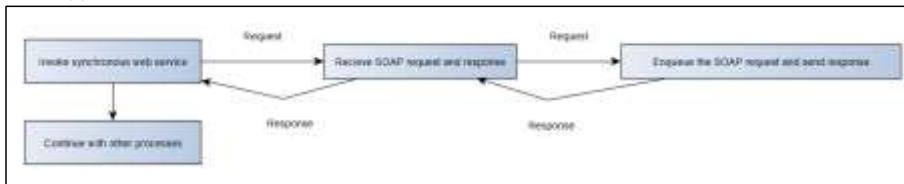
Understanding the impact of MTU sizes is important as previously stated there are issues which can be caused regarding the size of packets, particularly large packets, although there are benefits which do come alongside such as reducing processing power. IP Fragmentation by extension is important as if the programmer wants to avoid larger packets then it is a useful method to breakdown packets into fragments – this may be desired in case the MTU is smaller than the packet, or if increased chances of packet corruption want to be avoided – although there are security issues with IP fragmentation too, which should be taken into consideration before deciding if IP fragmentation should be done.

Commented [AG8]: Or, just keep data size down???

Synchronicity and Asynchronicity

Synchronicity

Synchronous web services distribute an instant response to queries. Clients wait until the server sends the response message back in the situation of a synchronous web service. An advantage of using a synchronous web service is that the status of the web service can be made known to the client application within a brief amount of time [46] [47] [48].

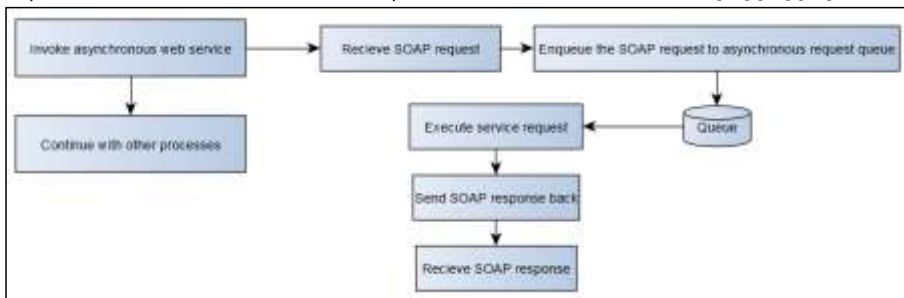


When a web service client sends a synchronous SOAP request to a service, the SOAP response will then be delivered back to the client when the process sees completion [49].

Asynchronicity

Asynchronous services need a noticeably higher amount of time compared to synchronous services to process requests made. Asynchronous services are advantageous in situations where the environment in use is being used for a service which may take a while to process single client requests, an example of this may be a loan processing service. The client who may invoke an asynchronous process can proceed with other items which need processing while waiting for a response [50] [51] [52].

A call-back pattern is a communication method used in asynchronous services. A response is sent to the requester when ready after said request is created to the service provider [53]. When a call-back is done without an acknowledgement pattern, a SOAP call-back header is needed if the contact information of the client isn't known at the time of deployment, and if the web service is asynchronous. Call-back headers allow clients to specify how to make contact during the time of a request for service – the client thus should provide a receive service or listener [49] [50] [53].



Upon completion of an asynchronous web server's request being delivered to a service the SOAP service is sent to the address of the client. This pattern does not need acknowledgement from the client, this is because it is a "fire-and-forget" pattern of message exchange for call-back [53] [54].

Applications for synchronicity

The request/reply message system used by PeopleSoft uses synchronous services, certain bits of code execute on the providing server while the client awaits a response. The client will send requests until a reply is received, this works differently to an asynchronous system such as an email

conversation where both parties need to reply back and forth in order for a conversation to take place [55].

API protocols such as REST are synchronous, as they handle requests synchronously and users are required to wait for a response from the API before they may carry on with using the application. More than one request cannot be processed at the same time, as requests taking a long time to be processed can severely bottleneck performance [56].

Applications for asynchronicity

Oracle's service delivery operation systems are asynchronous, examples of these include: USER_PROFILE, PERSON_BASIC_SYNC, DEPT_SYNC and DEPT_FULLSYNC (the sync means synchronize). These services are used to "push events" from one system to another, the contents of individual messages are not relevant and are often chosen to be ignored, the events in question are updates in components and batch processes [57] [58] [59].

MindMeister uses asynchronous services to share the visualisations of users to other users through the cloud in real time, allowing for collaborative work and coordinated task management. MindMeister delivers the mind maps created by users in an asynchronous manner [60].

Normal request processing for servlets need a single thread per request – thread exhaustion is common as threads may need to be blocked indefinitely and each one will need a request per client. Java EE uses asynchronous request processing to handle this issue as it is well-equipped to take care of long polling mechanisms [61] – examples of asynchronous request processing solutions include: Tomcat 6 Comet Processor [62], Jetty Continuations [63] and Glassfish Grizzly Connector [62].

Relevance to programmers

Choosing between synchronicity and asynchronicity for the type of response time is relevant to programmers as they both find different uses per what the programmer intends on creating. Synchronous services may be more desired if the end product is conversational, such as that of a chatroom, while asynchronous may be desired for more actively changing, collaborative works where items need to persistently be shared and receive frequent changes by multiple parties.

Interoperability

Interoperability is the ability of cooperation between different systems, components or products despite differences in execution, language or interface [28].

Syntactic interoperability refers to where two systems can communicate, while cross-domain interoperability refers to multiple parties working alongside each other, trading information. If multiple systems utilise the same data formats and communication protocols and communicate they are displaying syntactic interoperability, XML and SQL are examples of "common data formats and protocols", lower-level data forms provide for syntactic interoperability by guaranteeing that characters are kept in the same ASCII or Unicode format through every system which is communicating [29].

Semantic interoperability when systems trade data with a shared meaning, this is required when enabling a machine to compute logic as well as the ability to discover knowledge and inferencing. Semantic interoperability is concerned with syntax and the transmission of data with meaning – this is done through adding metadata which links data elements to a shared language. Syntactic interoperability acts as a prelude to semantic interoperability, finding syntactic errors, which allows

for receiving systems to request messages be re-sent if they are considered incomplete or uninterpretable – semantic interoperability is not possible if the syntax cannot represent the data trying to be sent [30] [31].

Interoperability in the context of software is used to elaborate on the abilities of different programs in exchanging data through common sets of exchange formats, and to interpret and write on the exact same file formats [32]. ISO/IEC-2382-01 [33] defines interoperability as “The capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units.” [34].

Java is an interoperable programming language, allowing programs to be written and then run with a Java Virtual Machine anywhere [35]. Programs written in Java have interoperability with all machines which can run a Java Virtual Machine of any implementation, whether it be Oracle, Android or IBM [36] - .NET languages and Common Language Runtime are interoperable in this manner also, using shared hosting virtual machines. HTML, CSS and JavaScript are interoperable as they work together in the functionality of webpages.

Methods for implementation of interoperability.

- Virtual machines are a “specialised intermediate language” which several different languages can be compiled to work within, this means languages using the same virtual machine can interoperate, this is because they will share a compiler and memory model, which in turn means a sole language can be used multiple times for others using the same virtual machine [35] [36] [37] [38]. Virtual machines can use type systems to guarantee accuracy of the languages which it can interpret [37].
- Foreign function interfaces allow programs written in a specific language to execute functions written in a different language. Most programming languages today have foreign function interfaces to the ‘C’ programming language [39], foreign function interfaces allow for wrapper libraries to be constructed, providing functionality from the library of a language in the host’s language [40] [41] [42].
- Object models are models which follow rules of standardisation, the use of object models is to allow for objects to have language-agnostic representation. Allowing objects to be used on several different programs and languages [44]. A challenge which exists with using object models however is differences between models, object orientated languages will try to pair data with code, but different every language will have its own method to pairing which will vary [45]. Design choices will not always map to different languages very well, classes which use a language that does not allow multiple inheritance will not translate well to a different language which does allow for multiple inheritance – defining a subset of a language which is compatible with the features of another language is a prominently utilised solution to this problem [43]. Examples of object models include ‘Common Object Request Broker Architecture’ (CORBA) and Component Object Model (COM).

Commented [AG9]: No discussion of protocols

Commented [AG10]: Not especially relevant

Commented [AG11]: 2862 words at this point.

References

1. Forouzan BA. *TCP/IP protocol suite*. McGraw-Hill Higher Education; 2002.
2. Hunt C. *TCP/IP network administration*. " O'Reilly Media, Inc."; 2002.
3. Callon RW. *Use of OSI IS-IS for routing in tcp/ip and dual environments*. RFC1195; 1990.

4. Dyer TD, Boppana RV. *A comparison of TCP performance over three routing protocols for mobile ad hoc networks*. In: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing; 2001.
5. Saldana J. *The effect of multiplexing delay on MMORPG TCP traffic flows*. In: 2014 IEEE 11th Consumer Communications and Networking Conference (CCNC); 2014.
6. Aracil J, Morato D. *Characterizing internet load as a non-regular multiplex of TCP streams*. In: Proceedings Ninth International Conference on Computer Communications and Networks (Cat. No. 00EX440); 2000.
7. Iren S, Amer PD, Conrad PT. *The transport layer: tutorial and survey*. ACM Computing Surveys (CSUR); 1999.
8. Khan IU, Hassan M. *Transport layer protocols and services*. International Journal of Research in Computer and Communication Technology; 2016.
9. Lopez JA, Sun Y, Blair PB, Mukhtar MS. *TCP three-way handshake: linking developmental processes with plant immunity*. Trends in plant science; 2015.
10. Hsu FH, Hwang YL, Tsai CY, Cai WT, Lee CH, Chang K. *TRAP: A three-way handshake server for TCP connection establishment*. Applied Sciences; 2016.
11. Dang F, Zhou P, Li Z, Liu Y. *NFC-enabled attack on cyber physical systems: A practical case study*. In: 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS); 2017.
12. Pshehotskaya E, Mikhalsky O, Vinnikov V. *The Simplified Two Generals Problem for Communication Immutability and Security Over Multiple Heterogeneous Channels*. Advanced Science Letters; 2018.
13. Lin YJ, Latchman H. *On the effects of maximum transmission unit in power line communication networks*. In: 2007 IEEE International Symposium on Power Line Communications and Its Applications; 2007.
14. Kodikara CK, Worrall ST, Kondo AM. *Optimal settings of maximum transfer unit (mtu) for efficient wireless video communications*. IEE Proceedings-Communications; 2015.
15. Murray D, Koziniec T, Lee K, Dixon M. *Large MTUs and internet performance*. In: 2012 IEEE 13th International Conference on High Performance Switching and Routing; 2012.
16. Li Y, Li D, Cui W, Zhang R. *Research based on OSI model*. In: 2011 IEEE 3rd International Conference on Communication Software and Networks; 2011.
17. Saxena P. *OSI reference model—a seven layered architecture of OSI model*. International Journal of Research; 2014.
18. Gilfeather P, Underwood T. *Fragmentation and High Performance IP*. In: IPDPS; 2001.
19. Atlas A. *Attacking ipv6 implementation using fragmentation*. Blackhat europe; 2012.
20. Bonica R, Baker F, Huston G, Hinden B, Troan O, Gont F. *IP fragmentation considered fragile*. RFC 8900; 2020.
21. Chandra DG, Kathing M, Kumar DP. *A comparative study on IPv4 and IPv6*. In: 2013 International Conference on Communication Systems and Network Technologies; 2013.
22. Huston G. *A tale of two protocols: IPv4, IPv6, MTUs and fragmentation*. The ISP Column; 2009.
23. Bonica R, Baker F, Huston G, Hinden B, Troan O, Gont F. *IP fragmentation considered fragile*. RFC 8900; 2020.
24. Kent CA, Mogul JC. *Fragmentation considered harmful*. ACM SIGCOMM Computer Communication Review; 1995.
25. Luckie M, Stasiewicz B. *Measuring path MTU discovery behaviour*. In: Proceedings of the 10th ACM SIGCOMM conference on Internet measurement; 2010.

26. Talak R, Karaman S, Modiano E. *Minimizing age-of-information in multi-hop wireless networks*. In 2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton); 2017
27. Correa C. Case 2. The SARS case. *IP fragmentation and patent pools*. Gene Patents and Collaborative Licensing Models: Patent Pools, Clearinghouses, Open Source Models and Liability Regimes; 2009
28. Wegner P. Interoperability. *ACM Computing Surveys (CSUR)*; 1996
29. Hosseini M, Dixon BE. *Syntactic interoperability and the role of standards*. In Health Information Exchange. Academic Press; 2016
30. Heiler S. *Semantic interoperability*. *ACM Computing Surveys (CSUR)*; 1995
31. Oukseil AM, Sheth A. *Semantic interoperability in global information systems*. *ACM Sigmod Record*; 1999
32. Zeng ML. *Interoperability*. KO Knowledge Organization; 2019
33. Neudecker C, Dogan M, Schlarb S. *D-OC5. 3 Evaluation Report* | OC5; 2010
34. Noura M, Atiquzzaman M, Gaedke M. *Interoperability in internet of things: Taxonomies and open challenges*. Mobile networks and applications; 2019
35. Venners B. *The java virtual machine*. Java and the Java virtual machine: definition, verification, validation; 1998
36. Breg F, Diwan S, Villacis J, Balasubramanian J, Akman E, Gannon D. *Java RMI performance and object model interoperability: Experiments with Java/HPC++*. Concurrency: Practice and Experience; 1998
37. Smith JE, Nair R. *The architecture of virtual machines*. Computer; 2005
38. Goldberg RP. *Architecture of virtual machines*. In Proceedings of the workshop on virtual computer systems; 1973
39. Furr M, Foster JS. *Checking type safety of foreign function calls*. *ACM SIGPLAN Notices*; 2005.
40. Yallop J, Sheets D, Madhavapeddy A. *A modular foreign function interface*. Science of Computer Programming; 2018
41. Furr M, Foster JS. *Checking type safety of foreign function calls*. *ACM Transactions on Programming Languages and Systems (TOPLAS)*; 2008
42. Paula GC. *A Foreign Function Interface For Pallene* (Doctoral dissertation, PUC-Rio); 2021
43. Gutknecht J. *Active Oberon for. NET: An Exercise in Object Model Mapping*. Electronic Notes in Theoretical Computer Science; 2001
44. Kim W, editor. *Modern database systems: the object model, interoperability, and beyond*. ACM Press/Addison-Wesley Publishing Co.; 1995
45. Chisnall D. *The Challenge of Cross-language Interoperability: Interfacing between languages is increasingly important*. Queue; 2013
46. Bultan T, Su J, Fu X. *Analyzing conversations of web services*. *IEEE Internet Computing*; 2016
47. Guermouche N, Godart C. *Timed model checking based approach for web services analysis*. In 2009 IEEE International Conference on Web Services; 2009
48. Bosworth A. *Developing web services*. In Proceedings 17th International Conference on Data Engineering; 2001
49. Chung S, Pan JR, Davalos S. *A special Web service mechanism: Asynchronous. NET Web services*. In Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services (AICT-ICIW'06); 2006
50. Wagener J, Spjuth O, Willighagen EL, Wikberg JE. *XMPP for cloud computing in bioinformatics supporting discovery and invocation of asynchronous web services*. *BMC bioinformatics*; 2009

51. Brambilla M, Ceri S, Passamani M, Riccio A. *Managing asynchronous web services interactions*. In Proceedings. IEEE International Conference on Web Services; 2004
52. Brambilla M, Guglielmetti G, Tziviskou C. *Asynchronous web services communication patterns in business protocols*. In International Conference on Web Information Systems Engineering; 2005
53. Zdun U, Voelter M, Kircher M. *Pattern-based design of an asynchronous invocation framework for web services*. International Journal of Web Services Research (IJWSR); 2004
54. Singh R, Mishra S, Kushwaha DS. *An efficient asynchronous mobile web service framework*. ACM SIGSOFT Software Engineering Notes; 2009
55. Kurtz D. *BEA Tuxedo: PeopleSoft's Application Server Technology*. In PeopleSoft for the Oracle DBA; 2012
56. Varga E. *Implementing Synchronous and Asynchronous REST APIs*. In Creating Maintainable APIs. Apress, Berkeley, CA; 2016
57. Friedman R, Mostefaoui A, Raynal M. *Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems*. IEEE Transactions on Dependable and Secure Computing; 2005
58. Arora NS, Redwood Shores CA. *Oracle® Streams for Near Real Time Asynchronous Replication*. Implementation, and Deployment of Database Replication; 2005
59. Hierons RM. *The complexity of asynchronous model based testing*. Theoretical Computer Science; 2012
60. Aoki K, Molnar P. *International collaborative learning using Web 2.0: Learning of foreign language and intercultural understanding*. In Global Learn. Association for the Advancement of Computing in Education (AACE); 2010
61. Gupta A. *Java EE 7 Essentials: Enterprise Developer Handbook*. "O'Reilly Media, Inc."; 2013
62. Rodríguez Pérez JA, Vera Gago J. *Learning environment based on social networks: EPSCCommunity* (Bachelor's thesis, Universitat Politècnica de Catalunya); 2010
63. Shen L, Yang C, Zhang F, Ren Y, Wu L, Zhu L. *A scalable Web GPS monitoring system based on Ajax push pattern*. In 2009 17th International Conference on Geoinformatics; 2009

Appendix