

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent of: Mohammed Boucadair

Reexamination Control No.: 90/015,315

U.S. Patent No.: 8,693,369

Reexamination Request Filed: October 23, 2023

For: METHOD OF ROUTING A DATA
PACKET IN A NETWORK AND AN
ASSOCIATED DEVICE

Examiner: Sorrel, Eron J

Technology Center/Art Unit: 3992

Attorney Docket No. 1127.0075

Attn: Mail Stop “*Ex Partes* Reexam”
Central Reexamination Unit
Office of Patent Legal Administration
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

**PATENT OWNER’S AMENDMENT AND RESPONSE TO FIRST OFFICE
ACTION**

In response to the first office action mailed on June 7, 2023 (the “Office Action”), patent owner Monarch Networking Solutions LLC (“Monarch” or “Patent Owner”), presents the following remarks and respectfully requests the rejections presented in the Office Action be withdrawn and patentability of all challenged claims be confirmed. None of the asserted prior art references, either alone or in combination, teach or render obvious the challenged claims of the ’369 Patent.

In addition, Patent Owner also submits an Amendment to the Claims. These new claims all are allowable for the same reasons as the independent claims. Additionally, the newly added claims recite additional features not present in the art of record.

If the USPTO determines that an additional fee is required, the Director is hereby authorized to charge any appropriate fees that may be required by this paper, and to credit any overpayment, to Deposit Account No. 50-4592, Attorney Docket No. 1127.0075.

TABLE OF EXHIBITS

Exhibit	Description
EX2001	RFC 791 ("Internet Protocol") (Sept. 1981), available at https://datatracker.ietf.org/doc/html/rfc791
EX2002	RFC 793 ("Transmission Control Protocol") (Sept. 1981), available at https://datatracker.ietf.org/doc/html/rfc793
EX2003	RFC 2131 ("Dynamic Host Configuration Protocol") (Mar. 1997), available at https://datatracker.ietf.org/doc/html/rfc2131
EX2004	RFC 1533 ("DHCP Options and BOOTP Vendor Extensions") (Oct. 1993), available at https://datatracker.ietf.org/doc/html/rfc1533

TABLE OF CONTENTS

I.	Amendments To The Clams.....	5
II.	Overview.....	11
A.	Background.....	11
B.	Claim Status.....	11
III.	'369 Patent Background.....	13
A.	State of the Art	13
1.	The Internet Protocol	13
2.	The '369 Patent.....	15
IV.	The Asserted References.....	17
A.	Biswas.....	17
B.	Fangman.....	20
V.	Claim Construction.....	21
VI.	Remarks	21
A.	Biswas Does Not Anticipate the Claims.....	21
1.	Biswas Does Not Teach Selecting An Identifier Based On The Port Mask (Claims 1, 3, 5, 6, 19, and 21)	21
2.	Biswas Does Not Teach A Configuration Message Including A Primary Address, A Port Mask, And A Routing Identifier (Claims 7, 10, 11, and 14).....	26
3.	Biswas Does Not Teach Assigning A Port Mask To A Terminal Equipment Connected To A Home Gateway (Claims 23-33).....	27
B.	Fangman Does Not Anticipate the Claims.....	29
1.	Fangman Does Not Teach The Use Of Port Masks (All Claims).....	29
2.	Fangman Does Not Route Packets Based On Port Ranges.....	30
3.	Fangman Does Not Teach The Configuration Message (Claims 7, 10, 11, and 14).....	31
VII.	Conclusion	35

I. AMENDMENTS TO THE CLAIMS

Below is the list of claims. No claims have been amended. New claims 18-32 have been added. Support for those claims can be found below. *Infra* § II.B.

LISTING OF CLAIMS:

1. (Original) A method of routing an IP data packet in a telecommunications network to a destination equipment, said packet comprising a primary destination address and a destination port number, said method comprising, on reception of said packet, the following steps:

determining a port mask assigned to the destination equipment, the port mask defining a range of port numbers to which the destination port number belongs;

selecting at least one identifier of the destination equipment of the packet from a plurality of equipment identifiers associated with said primary destination address based on the port mask; and

routing the packet to the destination equipment based on said identifier.
2. (Original) The method according to claim 1 of routing an IP data packet, wherein said at least one identifier is a secondary destination address of the destination equipment.
3. (Original) The method according to claim 2 of routing an IP data packet, comprising, before the routing step, a step of transforming the packet by adding said secondary address to the packet, and routing the packet based on said secondary destination address.
4. (Original) The routing method according to claim 1, wherein the at least one identifier is a route identifier specific to said destination equipment and the packet is routed based on said specific route identifier.

5. (Original) A device for routing an IP data packet in a telecommunications network to a destination equipment, said packet comprising a primary destination address and a destination port number, wherein said device comprises, on reception of said packet:
- a processor that determines a port mask assigned to the destination equipment, the port mask defining a range of port numbers to which the destination port number belongs;
 - the processor selects an identifier of the destination equipment of the packet from a plurality of equipment identifiers associated with said primary destination address based on the port mask; and
 - a communication unit that routes the packet to the destination equipment based on said identifier.
6. (Original) Telecommunications network equipment adapted to receive an IP data packet comprising a primary destination address and a destination port number, comprising at least one routing device comprising:
- a processor that determines a port mask assigned to a destination equipment, the port mask defining a range of port numbers to which the destination port number belongs;
 - the processor selects an identifier of the destination equipment of the packet from a plurality of equipment identifiers associated with said primary destination address based on the port mask; and
 - a communication unit that routes the packet to the destination equipment based on said identifier.
7. (Original) A method of configuring a piece of terminal equipment connected to a telecommunications network, comprising, on reception of a configuration request sent by said equipment, the following steps:

assigning to said piece of terminal equipment a primary address shared with a plurality of pieces of terminal equipment;

assigning to said piece of terminal equipment a port mask defining a range of numbers dedicated to it and a routing identifier; and

sending to said piece of terminal equipment a configuration message comprising said primary address, said port mask, and said routing identifier.

8. (Original) The method according to claim 7 of configuring a piece of terminal equipment, comprising a step of assigning to said piece of terminal equipment an identifier of a routing device adapted to route a data packet comprising said primary address as the destination address to said piece of terminal equipment, and the configuration message further comprising said routing device identifier.

9. (Original) The configuration method according to claim 7, wherein said configuration message is further sent to a routing device that routes to said piece of terminal equipment a data packet containing said primary address as the destination address.

10. (Original) A configuration server in a telecommunications network, wherein, on reception of a configuration request sent by a piece of terminal equipment, comprising:

a processor that assigns a primary address to said piece of terminal equipment, said primary address being shared with a plurality of pieces of terminal equipment;

the processor assigns to said piece of terminal equipment a port mask defining a range of numbers dedicated to it and a routing identifier; and

a communication unit that sends to said piece of terminal equipment a configuration message comprising said primary address, said port mask, and said routing identifier.

11. (Original) A method of configuring a piece of terminal equipment connected to a telecommunications network, wherein, on reception of a configuration message comprising configuration data comprising a primary address, a port mask defining a range of port numbers that is dedicated thereto, and a routing identifier, said method comprises the following steps:
- storing said configuration data; and
- before sending a data packet into the telecommunications network, selecting a port number in the range of port numbers defined by the port mask, said data packet comprising as source address the primary address of said piece of terminal equipment and as source port number the selected port number.
12. (Original) The configuration method according to claim 11, wherein, once an identifier of a routing device adapted to route a data packet comprising said primary address as the destination address to said piece of terminal equipment has been received, said method further comprises a step of sending the configuration data to said routing device.
13. (Original) The configuration method according to claim 11, wherein, on reception of a data packet comprising a destination address, a port number, and a routing identifier, said method further comprises a step of verifying that the port number belongs to the range of port numbers defined by the port mask, and a step of rejecting said packet if result of the verification is negative.
14. (Original) A piece of client terminal equipment connected to a telecommunications network, wherein, on reception of a configuration message comprising configuration data comprising a primary address shared with a plurality of pieces of terminal equipment, a port mask defining a range of port numbers that is dedicated to the piece of client

terminal equipment, and a routing identifier, the piece of client terminal equipment comprising:

a memory that stores said configuration data;

before sending a data packet in the telecommunications network, a processor that selects a port number in the range of port numbers defined by the port mask, said data packet comprising as source address the primary address of said piece of client terminal equipment and as source port number the selected port number.

15. (Original) A non-transitory computer readable medium encoded with program code instructions executable by a microprocessor to perform the method according to claim 1.
16. (Original) A non-transitory computer readable medium encoded with program code instructions executable by a microprocessor to perform the method according to claim 7
17. (Original) A non-transitory computer-readable medium encoded with program code instructions executable by a microprocessor to perform the method according to claim 11.
18. (New) The device for routing an IP data packet according to claim 5, wherein:
said identifier is a secondary destination address of the destination equipment.
19. (New) The device for routing an IP data packet according to claim 18, wherein:
the processor transforms the packet by adding said secondary address to the packet; and
the communication unit routes the packet based on said secondary destination address.
20. (New) The telecommunications network equipment according to claim 6, wherein said
identifier is a secondary destination address of the destination equipment.
21. (New) The telecommunications network equipment according to claim 20, wherein:
the processor transforms the packet by adding said secondary address to the packet; and
the communication unit routes the packet based on said secondary destination address.

22. (New) A non-transitory computer readable medium encoded with program code instructions executable by a microprocessor to perform the method according to claim 3.
23. (New) The method according to claim 1, wherein, the destination equipment is a terminal equipment connected to a home gateway.
24. (New) The method according to claim 3, wherein, the destination equipment is a terminal equipment connected to a home gateway.
25. (New) The device for routing an IP data packet according to claim 5, wherein, the destination equipment is a terminal equipment connected to a home gateway.
26. (New) The device for routing an IP data packet according to claim 19, wherein, the destination equipment is a terminal equipment connected to a home gateway.
27. (New) The telecommunications network equipment according to claim 6, wherein, the destination equipment is a terminal equipment connected to a home gateway.
28. (New) The telecommunications network equipment according to claim 21, wherein, the destination equipment is a terminal equipment connected to a home gateway.
29. (New) A non-transitory computer readable medium encoded with program code instructions executable by a microprocessor to perform the method according to claim 24.
30. (New) The method according to claim 7, wherein, the piece of terminal equipment connected to a telecommunications network is connected to a home gateway.
31. (New) The configuration server according to claim 10, wherein, the piece of terminal equipment is connected to a home gateway.
32. (New) The method according to claim 11, wherein, the piece of terminal equipment connected to a telecommunications network is connected to a home gateway.

33. (New) The piece of client terminal equipment connected to a telecommunication network according to claim 14, wherein, the client terminal equipment is connected to a home gateway.

II. OVERVIEW

A. Background

On October 23, 2023, D. Joseph English requested *ex parte* reexamination (the “Request” of U.S. Patent No. 8,693,369 (the “’369 Patent”) on behalf of Unified Patents, LLC (“Unified” or “Requester”). The Request challenged claims 1-3 and 5-17 of the ’369 Patent (the “Challenged Claims”) based primarily on U.S. Patent Nos. 7,624,195 (“Biswas”) and 8,363,647 (“Fangman”). The Office ordered reexamination of all claims on January 18, 2024 (*sua sponte* including claim 4) and issued rejections of all claims in a June 7, 2024 non-final office action. All Claims were rejected under pre-AIA 35 U.S.C. § 102(e) as anticipated by Biswas. Claims 1-8, 10, 11, and 13-17 were rejected under pre-AIA 35 U.S.C. § 102(b) as anticipated by Fangman.

Monarch respectfully requests that the Office withdraw all rejections to the claims and confirm the patentability of the ’369 Patent in light of the following remarks. Specifically, neither reference discloses or suggests all elements of the claims.

B. Claim Status

Issued claims 1-17 have not been amended. For the reasons that follow, Monarch respectfully requests reconsideration and a confirmation of allowance of the previously issued claims.

The newly added dependent claims 18-21 seek to incorporate limitations found in claims 2 and 3 in claims that depend from independent claims 5 and 6. As such, claims 18-21 are necessarily narrower than claims 5 and 6. Claim 22 depends from claim 3 and narrows the claim to “a non-transitory computer readable medium...” Support for claims 18 and 20 can be found

in claim 2. Support for claims 19, and 21 can be found in claim 3. Support for claim 22 can be found in claims 3 and 15.

Additionally Support for claims 18 and 20 can also be found in the '369 Patent at 3:48-57 (explaining that the identifier may be a secondary address), and 9:9-17 (in one embodiment, the routing identifier is a private address).

Support for claims 19, and 21 can be found in the '369 Patent at 3:58-64 (explain that the packet may be transformed or encapsulated with the secondary address and routed based on the secondary address), and 12:58-13:4 (describing several ways to transform or encapsulate the packet with a secondary private address, and route the packet using that address).

Support for claim 22 can be found in the '369 Patent at 4:40-57 (explaining that the invention provides a computer-readable data medium containing instructions for the computer program described in the specification).

Newly added claims 23-33 depend from other claims and add features related to terminal equipment connected to a home gateway. Support for these claims can be found throughout the specification. For example, in one embodiment, the '369 Patent describes that the destination equipment may be terminal equipment and a plurality of terminal equipment may be connected to a home gateway, as illustrated in Figure 3:

A telecommunications network RT of an operator on which a service provider has deployed an IP connectivity or access service architecture is described with reference to FIG. 3. A plurality of pieces of terminal equipment have access to this network. Below the term terminal equipment refers equally to the pieces of terminal equipment **110, 120, 130, 140, 160, 170**, for example telephones or personal computers, connected directly to the network RT and to a home gateway **230, 220** between a home network and the telecommunications network RT.

'369 Patent at 8:1-10. Other sections of the '369 Patent support that the terminal equipment may be connected to a home gateway. *See, e.g., id.* at 1:45-51, 2:4-7, 3:30-34.

III. '369 PATENT BACKGROUND

The '369 Patent provides solutions to enable routing data packets across domains by assigning port masks to terminal device, preserving public IP addresses and allowing reduced overhead when translating packet headers. *See* '369 Patent at 2:29-64.

A. State of the Art

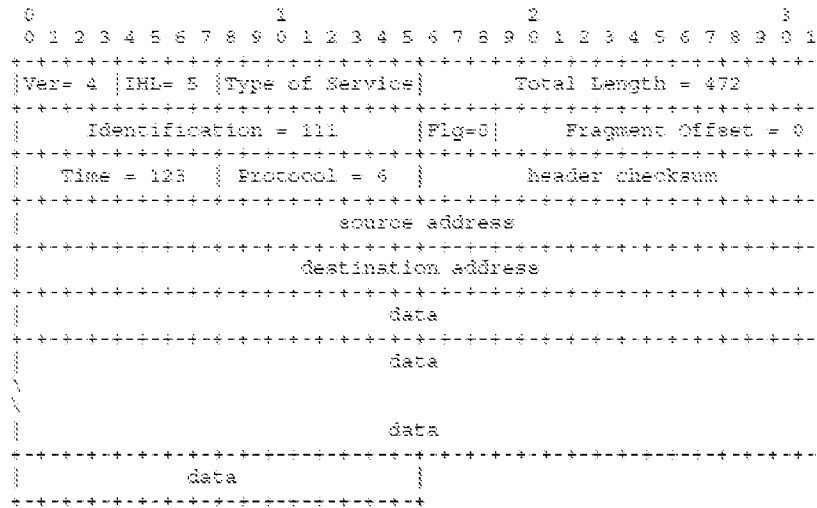
1. *The Internet Protocol*

The Internet allows packets of data to be transmitted between remote devices or networks. These packets are sent and received using Internet Protocol ("IP"), and traditionally, Internet Protocol version 4 ("IPv4"). *See* EX2001 at 1; '369 Patent at 1:15-18, 1:53-54.

IPv4 defines numerous aspects of the Internet Protocol, including the structure of the IP packets. *See generally* EX2001. These packets include a "header" part that carries the information used to transmit data packets to their destinations. EX2001 at 8, 11. The IP header uses "addresses" to identify the source and destination of the packet, typically in the form of 32-bit numbers. EX2001 at 1-2, 11, 14.

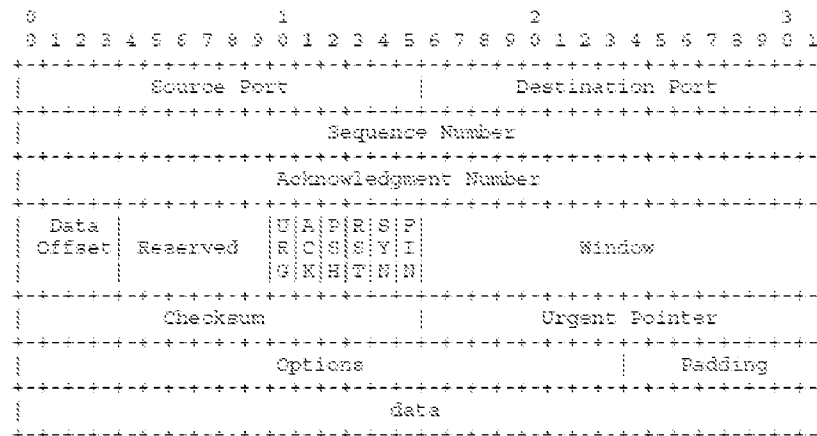
The IPv4 packet also includes a data or payload section carrying the data being transmitted between devices. *Id.* at 1. Within the data section, the packet often includes a higher layer protocol such as Transmission Control Protocol ("TCP") or User Datagram Protocol ("UDP"). *Id.* at 1, 5. These protocols include their own header information. EX2002 at 15. In the case of TCP and UDP, the header includes a source port and a destination port. *Id.* Like IPv4 addresses, the port numbers are typically represented as 16-bit numbers. *Id.*

The resulting structure of an IPv4 packet containing a TCP packet is illustrated below (where the TCP packet would be within the data portion of the IPv4 packet):



Example Internet Datagram

Figure 6. (RFC 791)



TCP Header Format

Note that one tick mark represents one bit position.

Figure 3. (RFC 793)

EX2001, Fig. 6, EX2002, Fig. 3.

In the IP protocol, unique addresses are used to identify individual devices on a network. See '369 Patent at 1:35-59 (noting the risk of exhaustion of addresses and the increased number of addresses available in IPv6). So any device directly connected to a public IP network, such as the Internet, needs a unique IP address to communicate with the rest of the network. See *id.* (same). By the time of the '369 Patent, IP practitioners understood that the internet would run out

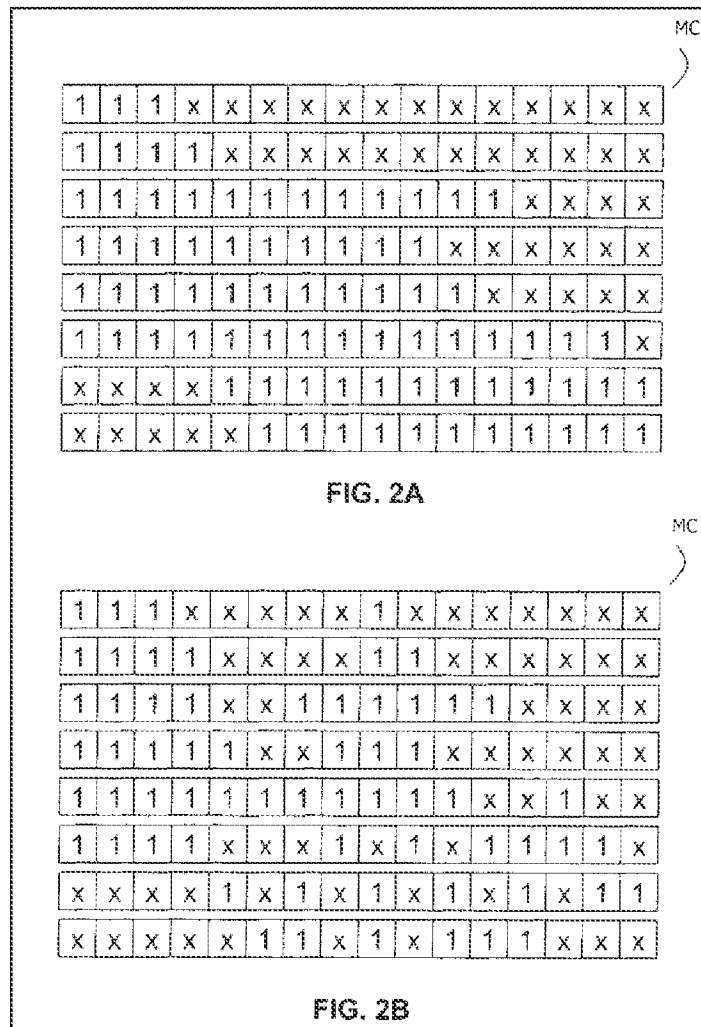
of unique addresses. '369 Patent at 1:53-54. Those practitioners sought solutions to either avoid exhaustion or replace IPv4 addresses. *Id.* at 1:53-54.

One solution is Internet Protocol version 6 ("IPv6"). *Id.* at 1:55-59. IPv6 dramatically increases the available number of addresses by using larger numbers to represent the address, typically represented as a 128-bit value. *Id.* at 1:55-59. However, at the time of the '369 Patent, IPv6 adoption was proceeding slowly. *Id.* at 1:59-60. Until IPv6 is more widely and universally adopted, other solutions are needed to conserve public IPv4 addresses. *See, e.g., Id.* at 1:66-2:1 (referencing other solutions taken to address the imminent risk of IPv4 address exhaustion).

One alternative solution is Operator NAT, which maintains "state tables" associating private IP addresses and port numbers with public IP addresses and port numbers. *Id.* at 2:19-41. But Operator NAT has several disadvantages, such as significantly increasing the processing overhead required to route packets. *Id.* at 2:19-41.

2. *The '369 Patent*

Mohamed Boucadair, the inventor of the '369 Patent, sought to alleviate these drawbacks while working at Orange, a French telecommunications company. *Id.* at cover, 1:53-54, 2:45-55. One solution described by the '369 Patent is to provide a shared public address and a portion of the available public port numbers to terminal devices connected to a private network. *Id.* at 3:11-29. In this embodiment, the '369 Patent describes assigning and transmitting port masks from an edge router to a terminal device. *Id.* at 7:29-32. These port masks represent a range or set of port numbers. *Id.* at 7:49-67. Examples of port masks are illustrated in Figs. 2A and 2B:



By conveying and maintaining the port range assignments as masks, edge routers using the methods of the '369 Patent can significantly reduce overhead. *Id.* at 2:19-50. For example, this technique allows the terminal devices to use public IP addresses and a set (or range) of port numbers when transmitting a packet. *Id.* at 4:65-67, 7:32-35. This way, the edge router does not need to perform NAT on the packet before sending it into the public network, simplifying packet routing. *See id.* at 2:15-22, 3:18-23, 8:57-67 (similar).

The '369 Patent has seven independent claims that fall into roughly two categories. The first group of independent claims are 1, 5, and 6. These claims are directed to methods and devices that route a packet to a destination device based on determining a port mask assigned to

the destination device. The second group includes independent claims 7, 10, 11, and 14. These claims are directed to methods and devices that assign a port range to a device in the form of a mask and transmitting the mask to the device.

IV. THE ASSERTED REFERENCES

A. Biswas

Biswas is U.S. Patent No. 7,624,195, titled “Method and apparatus for distributed network address translation processing.” Biswas at cover. It describes methods for assigning global addresses and port ranges to a group of devices and using ports to distinguish messages from separate devices. *Id.*, Abstract.

Like the '369 Patent, Biswas recognized problems posed by the imminent exhaustion of public IPv4 addresses. *Id.* at 1:14-27. Biswas also recognized the drawbacks of existing address conservation techniques, such as NAT. *Id.* at 3:38-41. To avoid these drawbacks, Biswas describes a method to distribute NAT functions across a private network. *Id.* at 4:15-20.

Biswas invention is described in reference to its Figure 1:

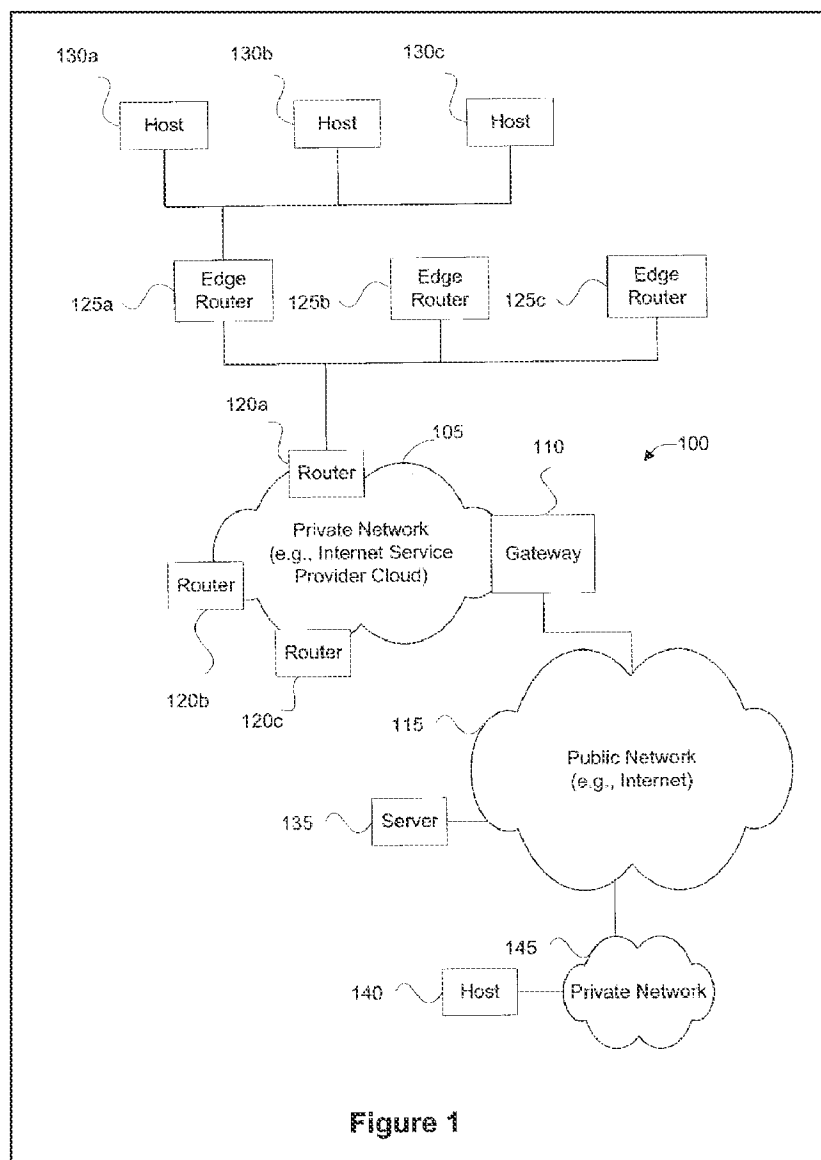


Figure 1

Biswas, Fig 1. As shown above, Biswas describes a private network 105, such as an internet service provider (“ISP”), connected to a public network 115, such as the Internet, via gateway 110. *Id.* at 3:42-46. Private network 105 includes a number of “intermediary hubs,” in this case routers 120a-c. *Id.* at 3:46-48. Routers 120a-c are aggregators that provide connectivity for edge routers 125a-c. *Id.* at 3:48-50. Edge routers 125a-c, in turn, are customer premises equipment (“CPE”), such as a modem, that provide network access for hosts 103a-c. *Id.* at 50-53. Biswas discussion focuses on aggregators, CPEs, and PCs. *Id.* at 56-59.

Biswas describes techniques that distribute NAT functions—which were previously performed by gateway 110—to routers 120a-c and, potentially, edge routers 125a-c. *Id.* at 4:14-20. The preferred embodiment describes enabling downstream devices to perform NAT by assigning those devices two IP addresses, an “inside” address and an “outside” address. *Id.* at 3:66-4:3 (describing inside addresses), 4:26-28 (noting that devices need to be informed of the outside address). However, Biswas notes that this creates several challenges, and proposes solutions to those problems. *See id.* at 4:21-29.

In a preferred embodiment, an outside address is directly correlated to an inside address, allowing simple, header-only NAT. 44:41-50. In another embodiment, multiple inside devices share the same outside address. *See id.* at 4:51-60. There, the gateway may use port ranges to differentiate the aggregators when there are not enough outside IP addresses available to be uniquely assigned to the aggregators. *Id.* at 4:51-5:8.

Biswas also describes techniques for forwarding packets within the network. A preferred solution is injecting the outside addresses into the routing tables of devices on the private network. *Id.* at Fig. 2, 5:52-58, 6:15-32. This allows packets with the outside address to be routed within the private network. *See id.* at 6:15-32. Alternatively, tunneling can route packets within the private network. *Id.* at 6:37-7:19. The tunneling embodiment uses simple header only NAT to forward encapsulated packets. *Id.* at 6:42-46.

Biswas’s two embodiments do not combine all the limitations of the claims of the ’369 Patent. In the first embodiment, port numbers are not used to identify a routing path for a packet. In the second embodiment, port numbers are used in lieu of an identifier of the destination. Biswas does not teach a combination of these two distinct embodiments. Moreover, Biswas does not describe assigning port ranges to terminal equipment (e.g., Biswas’s PCs and hosts 130a-c).

Instead, Biswas assigns a port range only to routing devices in the network, specifically the aggregators (routers 120a-c) and the CPEs (edge routers 125a-c).

B. Fangman

Fangman is U.S. Patent No. 8,363,647, titled “System and method for configuring an IP telephony device.” Fangman at cover. It describes systems for providing DHCP lease information to an IP telephone to enable communications between an IP telephone and another device. *Id.* at Abstract.

Fangman focuses on voice over IP technologies (“VoIP”). *Id.* at 1:18-22. Fangman explains that the VoIP market faced significant technical challenges, including problems caused by the limited number of public IP addresses, NAT, and firewalls. *Id.* at 1:35-2:23. One of Fangman’s proposed improvements is a system that provides setup information to an IP telephone that assigns an available port range to an IP telephone. *Id.* at 3:7-33. Once the port range is assigned, the IP telephone can communicate using one of the available ports. *Id.* at 3:60-65.

When a packet is transmitted, Fangman describes using an alternate version of NAT called “Network Address Persistent Port Translation” or “NAPPT.” *Id.* at 3:66-4:11. In NAPPT, the private source address is translated to a public address, but the source port is unchanged. *Id.*, 4:6-11. The public source address and the “persistent” port can identify the device corresponding to a packet sent back to the private network. *Id.* at 4:12-27.

Notably, Fangman does not teach routing a packet based on determining a port range (or port mask) for a packet received at a router. Instead, Fangman only uses the port number in the packet to identify a destination device in NAPPT, similar to traditional NAT or PAT. *Id.* at 17:60-18:10. Fangman also does not mention the use of port masks in its NAPPT process.

V. CLAIM CONSTRUCTION

In a Reexamination proceeding, the claims of the '369 Patent must be given their broadest reasonable interpretation in light of the specification as it would be understood by one of ordinary skill in the art:

The Patent and Trademark Office (“PTO”) determines the scope of claims in patent applications not solely on the basis of the claim language, but upon giving claims their broadest reasonable construction “in light of the specification as it would be interpreted by one of ordinary skill in the art.” *In re Am. Acad. of Sci. Tech. Ctr.*, 367 F.3d 1359, 1364 (Fed. Cir. 2004). Indeed, the rules of the PTO require that application claims must “conform to the invention as set forth in the remainder of the specification and the terms and phrases used in the claims must find clear support or antecedent basis in the description so that the meaning of the terms in the claims may be ascertainable by reference to the description.” 37 C.F.R. § 1.75(d)(1). It is therefore entirely appropriate for a court, when conducting claim construction, to rely heavily on the written description for guidance as to the meaning of the claims.

Phillips v. AWH Corp., 415 F.3d 1303, 1316-17 (Fed. Cir. 2005) (en banc); *see also In re Smith Int'l, Inc.*, 871 F.3d 1375, 1381-83 (Fed. Cir. 2017) (“The correct inquiry in giving a claim term its broadest reasonable interpretation...is an interpretation that corresponds with what and how the inventor describes his invention in the specification, *i.e.*, an interpretation that is consistent with the specification.” (internal quotes and citations omitted)).

The Office Action appears to give all terms their plain and ordinary meaning and does not appear to propose any particular interpretation of any limitation.

VI. REMARKS

A. Biswas Does Not Anticipate the Claims

1. *Biswas Does Not Teach Selecting An Identifier Based On The Port Mask (Claims 1, 3, 5, 6, 19, and 21)*

Biswas does not anticipate the claims claiming an identifier of a destination equipment is selected based on a port mask or port range. Claims 1, 5, and 6 each claim routing or receiving a packet that includes a destination port number. The claims require that a port mask be

determined, the port mask defining a port range to which the destination number belongs. That port mask is assigned to a destination equipment. Based on identifying the port mask, the packet is routed to the destination equipment using an identifier of the destination equipment. Claims 3, 19, and 21 add that routing the packet further includes transforming the packet by adding the identifier (a secondary address of claims 2, 18, and 20) to the packet.

a) The Relevant Embodiment of Biswas Routes Packets Based On Port Numbers, Not Ranges

The Office Action states that the “destination equipment” of claims 1, 5, 6 corresponds to Biswas’s router 120a (the aggregator). Office Action at 4, 8, 11.¹ However, Biswas does not route packets to the aggregator based on a destination port mask or range. Instead, Biswas teaches routing such packets based, in part, on the port *number*, using traditional NAT techniques.

The key aspect of Biswas’s invention is assigning corresponding outside addresses and inside addresses to aggregator devices (routers 120a-c) to reduce the NAT functions at the gateway. Biswas at 4:21-40. As such, Biswas’s discussion of port ranges is limited. To the extent Biswas describes port ranges, it describes an alternate embodiment that assigns port ranges to devices in the private network along with an outside address. *Id.* at 4:51-60. This allows a packet with an outside address to be routed to the correct device without performing NAT at the higher level devices, such as the gateway. *See id.* at 4:21-23. However, this discussion of assigning port ranges does not directly address packet handling (i.e., how a port number is identified or an address selected).

¹ The Office Action does not explicitly identify the destination equipment of claim 6, but it appears to be the aggregator like claims 1 and 5. *See* Office Action at 11 (citing Biswas at 4:56-60).

Where Biswas does address packet handling, Biswas explicitly teaches routing a packet based on the port *number*, not the port *range*. Specifically, Biswas states that “When the gateway 110 receive[s] an outside packet, it could then be routed to a selected aggregator based on both the outside address and the port *number*.” *Id.* at 5:6-8; *see also* Office Action at 6 (citing same). This is similar to the prior art NAT process described by the ’369 Patent, “which [] associates the private IP address and the port associated with the terminal with [a] public IP address and port...” ’369 Patent at 1:45-52.

In traditional NAT, a source port number is directly associated with a public source address when a packet is sent out of the network. *Id.* Therefore, when a return packet is received, the NAT device compares the destination port number to the previously identified source port number and modifying the destination address accordingly. Biswas does not teach modifying this system and, instead, appears to rely on it. *E.g.*, Biswas at 4:44-50. Therefore, Biswas does not teach determining a port mask, because the Biswas uses the prior art method of a one-to-one association of a port number to a destination device.

Even if this embodiment taught using a port mask, it does not teach routing a packet based on a selected identifier. The Office Action contends that the destination address of claims 1, 5, and 6 is Biswas’s outside address, Office Action at 4-5, and the claimed identifier is an inside address. Office Action at 5-6. But Biswas teaches that, when a port number is used, the packet is routed “based on [] the *outside* address...” Biswas at 5:6-8.

As noted above, a key aspect of Biswas is injecting the outside address into the private routing table, allowing packets to be routed using the outside address and avoiding NAT at the higher level devices. *Id.* at 4:14-23, 6:19-32. Because Biswas modifies its private network to use the outside address, Biswas does not teach that a separate identifier of the destination equipment

is needed or used to route the packet.² Therefore, Biswas does not teach the claimed identifier of the destination equipment.

b) The Other Embodiments Of Biswas Highlight The Difference Between Biswas And the Claims Of The '369 Patent.

The other sections of Biswas cited in the Office Action further illustrate the difference between Biswas and the '369 Patent. For example, with respect to the step of “selecting at least one identifier of the destination equipment,” the Office Action points to Biswas’s discussion of tunneling packets from the gateway to the aggregator. Office Action at 5-6 (citing Biswas at 6:42-48). This embodiment uses “very basic header-only NAT,” *i.e.*, without relying on port numbers found in the payload and directly associating an inside address with an outside address. *Id.* at 6:42-46. This is the same approach taught in the preferred embodiment. *See* Biswas at 6:19-32 (modifying routing tables to associate both an outside address and an inside address with one device). The tunneling embodiment does not contemplate the use of port numbers, at all. Indeed, using a port number in this embodiment would defeat the invention by moving the full NAT functions back to the gateway.

Another example is Biswas’s discussion of subnetting. *See* Office Action at 5 (citing Biswas at 7:42-47). The Office Action’s reliance on this embodiment presents two problems. *First*, this embodiment describes a CPE, rather than the aggregator relied on in the Office Action for the destination equipment limitation. This embodiment discusses packet routing from the

² For this reason, this embodiment also does not teach Claims 3, 19, and 21, which requires that the packet is transformed by adding a secondary address to the packet. As explained in the next paragraph, the embodiment describing encapsulation does not rely on port numbers at all, much less port ranges. Biswas does not teach combining its embodiments, and therefore, does not anticipate claims 3, 19, and 21.

aggregator to the CPE, not from the gateway to the aggregator. Biswas at 7:23-26 (“As with the aggregator, the CPEs would need to be both alerted to outside packets requiring NAT functions and have access to the outside addresses and/or port ranges in order to perform the NAT functions.”). The Office Action does not explained how the teaching of these separate embodiments could be combined.

Second, the discussion of subnets in this embodiment does not teach that a packet could be routed using on an identifier identified selected based on the port mask. Instead, this embodiment suggests the aggregator routes a packet to a CPE based only on the outside address and the port number, consistent with the preferred embodiment discussed above. *See* Biswas at 5:6-8. For example, Biswas explains that “the aggregator would act as a GAPP/RAPID server from the CPE side...” Biswas at 7:38-41. Biswas earlier explains that “the GAPP/RAPID server is responsible for assigning outside address and/or port ranges,” but “[t]he inside address is generally assigned using DHCP.” *Id.* at 5:29-31. Thus, by using GAPP/RAPID, Biswas teaches that, when a port number is used, the packet is routed based on the outside destination address (not an identifier), and NAT is distributed from the aggregation device to the CPE, consistent with the goals of Biswas. *See id.* at 4:15-20 (“[I]n the present invention as illustrated in FIG. 1 the gateway **110** is not solely responsible for these NAT functions. Instead, the responsibility is distributed between the gateway **110**, the intermediary routers **120 a**, **120 b** and **120 c** and potentially the edge routers **125 a**, **125 b** and **125 c**.”).

Accordingly, no embodiment of Biswas teaches both the port mask and routing identifier elements of claims 1, 3, 5, 6, 19 and 21.

2. *Biswas Does Not Teach A Configuration Message Including A Primary Address, A Port Mask, And A Routing Identifier (Claims 7, 10, 11, and 14)*

Claims 7, 10, 11, and 14 each include a configuration message or configuration data that includes, 1) a primary address, 2) a port mask, and 3) a routing identifier.

Turning first to the port mask, Biswas never describes or suggests sending or receiving a port mask in a configuration message. The Office Action points to a reply sent from an aggregator to a CPE. Office Action at 14, 16 (citing Biswas at 7:23-37), 17-18, 20 (citing Biswas at 2:60-65). But these sections only discuss sending a port range to a device, not a port mask.

Biswas also fails to teach a single message comprising the three claimed elements (the 1) a primary address, 2) a port mask, and 3) a routing identifier). With respect to claims 7 and 10, the Office Action points to a discussion of the aggregator providing certain information to the CPE:

As with the aggregator, the CPEs would need to be both alerted to outside packets requiring NAT functions and have access to the outside address and/or port-ranges in order to perform the NAT functions. Therefore, just as the aggregators would be aware of their outside addresses and/or port-ranges, so would the CPEs. When a CPE requests an inside address, the aggregator (or some device in concert with the aggregator) would reply with an inside DHCP address and the outside address and/or port-ranges

Office Action at 14, 17 (citing Biswas at 7:23-37). Though this discussion mentions a reply with an inside DHCP address, and an outside address and/or port-ranges, it does not teach sending all the in a single message.

To the contrary, this embodiment necessarily teaches multiple messages, first a DHCP offer and then an reply with the outside address and/or port ranges. DHCP is defined by RFC 2131 and includes an options portion that is further defined by RFC 1533. *See generally* EX2003; EX2004. In DHCP, a client device sends a DHCP discover message to obtain configuration parameters from a server. EX2003 at 6, 13. The server responds with a DHCP offer

message that includes the requested parameters. *Id.* at 13. The DHCP offer message includes a set of parameters defined by RFC 2131, but none of these include an identifier (distinct from the primary address), or a port range. *See id.* at 10.

Thus, Biswas's aggregator must send at least two messages in order provide the inside DHCP address, and an outside address and/or port-ranges to the CPE, one of which is the DHCP message.

With respect to claims 11 and 14,³ the Office Action points to two separate messages received at an edge router, an assignment and a notification:

The edge router then receives [1] an assignment of the inside address, a range of useable ports and [2] a notification of an outside address that outside devices will use to communicate with the responsible devices.

Office Action at 18 (annotated) (citing Biswas at 2:60-65). This description aligns with Biswas's detailed description of the separate DHCP message and outside address/port range message. Just like claims 7 and 10, Biswas's edge router must receive at least two messages in order provide the inside DHCP address, and an outside address and/or port-ranges. Accordingly, Biswas does not teach the claimed configuration message.

3. *Biswas Does Not Teach Assigning A Port Mask To A Terminal Equipment Connected To A Home Gateway (Claims 23-33).*

Biswas does not anticipate the claimed assignment of a port mask to a terminal equipment connected to a home gateway. Each of claims 23-33 includes a version of this limitation. But, because Biswas only assigns port ranges to intermediate devices and routers, it does not anticipate these claims of the '369 Patent.

³ The Office Action does not cite support for the configuration message of claim 14, and therefore the validity of claim 14 should be affirmed because no teaching in Biswas discloses that element.

Claims 23-33 of the '369 Patent, on the other hand, describes and claims assigning port ranges to terminal equipment connected to a home gateway. *See also* '369 at 3:30-34. As a result, these claims include: 1) a port mask assigned to a terminal equipment connected to a home gateway, and 2) that the mask defines a range of numbers assigned to the terminal equipment.

On the other hand, Biswas describes functions at a gateway, aggregation devices, and CPE devices. Each of these devices are intermediate devices that do not teach a terminal equipment connected to a home gateway. For example, Biswas explains that the aggregator is an “intermediary hub” that provides connectivity to another set of edge routers. Biswas at 3:46-50. The edge routers then provide network access to a further set of hosts. *Id.* As explained above, Biswas focuses on assigning addresses to these routing devices in the network. This is a different concept from assigning port ranges to terminal equipment connected to a home gateway as claimed by these claims of the '369 Patent.

Biswas implicitly recognizes that it does not teach the terminal equipment connected to a home gateway of claims 23-33 of the '369 Patent. In the brief discussion of the function of the PCs in Biswas's system, Biswas explains that a CPE “could either perform the NAT functions itself, or...inform[] the PCs of a global address and [the CPE's] own port range.” Biswas at 7:59-62. But Biswas never states (or even suggests) that a port range would be assigned to the PC rather than the CPE. Biswas also disparages this approach, stating that it “may not always be feasible.” *Id.* at 7:64-67.

In sum, Biswas's specification focuses on distributing NAT functionality amongst a group of routers. It does not suggest that its teaching could apply to destination or terminal equipment. Accordingly, Biswas does not anticipate claims 23-33 of the '369 patent.

B. Fangman Does Not Anticipate the Claims

1. Fangman Does Not Teach The Use Of Port Masks (All Claims)

Every independent claim of the '369 Patent claims a port mask that defines a range of port numbers. Fangman does not describe port masks at all and cannot anticipate the claims.

The '369 Patent's invention "is based on an entirely novel and inventive approach to routing a data packet that relies on a unique combination consisting of the destination address of the packet, the port mask defining a range of port numbers that includes the destination port number of the received packet and a routing identifier of the destination terminal equipment." '369 Patent at 3:11-16. The invention provides numerous advantages, one of which is a solution to IPv4 address exhaustion without significant capital investment. *Id.* at 3:17-29.

The '369 Patent explains that the port mask is one aspect that provides the benefits of the invention:

The invention consists in reducing the range of port numbers available to terminal equipment for each application by assigning a piece of terminal equipment a port mask that defines a sub-portion of this range. The port mask assigned in this way to a particular piece of terminal equipment is used in combination with its primary IP address to route data packets that are sent to it without degrading IP service.

'369 Patent, 7:29-35.

In contrast, Fangman is entirely silent as to port masks. The Office Action points to discussion in Fangman related to a destination port number and a range of port numbers. Office Action at 23, 26, 27 (citing Fangman at 17:60-18:9), 29, 31-32 (citing Fangman at 8:30-33, 11:48-50), 35-36 (citing 3:26-33, 3:50-56); *see also* Office Action at 33 (not identifying any part of Fangman as disclosing the port mask found in the preamble of claim 11). But, none of these sections describe or suggest a port mask.

Accordingly, Fangman does not teach any of the independent claims of the '369 Patent.

2. *Fangman Does Not Route Packets Based On Port Ranges*

Even if Fangman taught the use of port masks, Fangman does not teach using port ranges to route packets to a destination device, as required by claims 1–5, and 6 of the '369 Patent.

Independent claims 1, 5, and 6 claim, in part, a process where a router receives a packet and determines whether or not the destination port number in the packet falls within a ranges of port numbers assigned a terminal device connected to the router. The determination is made using port masks that correspond to the assigned ranges. Specifically, element 1[a] recites “determining a port mask assigned to the destination equipment, the port mask defining a range of port numbers to which the destination port number belongs.” '369 Patent at 14:20-22. Claim element 1[b] also requires “selecting at least one identifier of the destination equipment of the packet . . . based on the port mask”. *Id.* at 14:23-26. Claims 5 and 6 include similar limitations from the perspective of “a device for routing an IP data packet . . .” and “telecommunications network equipment . . .” *Id.* at 14:46-53, 14:60-67.

Fangman's disclosure focuses on the initial setup of an IP telephony connection. Fangman at 16:29-36, 16:53-57. Fangman explains that a service gateway can assign a range of ports to an IP telephone (i.e., a terminal device). *Id.* (discussing port allocation in the context of a DHCP process). But Fangman only briefly discusses handling packets received at the service gateway. This discussion is quoted below:

Conversely, when the SG 170 receives a data packet from the IP device destined for the IP telephone 120, the data packet may include a public destination IP address, a destination port number; and source information associated with the IP device, where the destination port number is in the assigned range of port numbers for the IP telephone 120. In one embodiment, the public destination IP address and the destination port number may be used to uniquely identify the IP telephone 120. Performing a NAPPT on the data packet may include using the public destination IP address and the destination port number to uniquely identify the IP telephone 120, and changing the public destination IP address to a private destination IP address while leaving the destination port number unchanged. Note that in this embodiment, the private IP address may be an IP address of the IP

telephone 120. After NAPPTing the data packet, the SG 170 may send the data packet to the IP telephone 120.

Id. at 17:60-18:9. This is the same section cited in the Office Action for the “determining” and “selecting” elements.

Above, Fangman first describes the contents of a packet received at service gateway 170 (i.e., that it may include “[1] a public destination IP address, [2] a destination port number; and [3] source information associated with the IP device, where the destination port number is in the assigned range of port numbers for the IP telephone 120.”). *Id.* at 17:62-65. Fangman then explains how this packet may be handled: “In one embodiment, the public destination IP address and the destination port number may be used to uniquely identify the IP telephone 120.” *Id.* at 17:65-18:1. Focusing on the destination port number, Fangman teaches that the packet includes a port number within the range assigned to the destination IP telephone. *Id.* at 17:50-59. The port number may be used to identify the destination IP telephone. *Id.*

This is part of Fangman’s Network Address Persistent Port Translation (“NAPPT”), and it relies on a “persistent” port number to transmit packets between networks. *Supra* § IV.B. Using this technique, the service gateway checks whether the persistent port number in the destination packet matches the port number used by the IP telephone in a prior outgoing transmission. *Id.* at 17:46-50. But by using a single persistent port *number*, NAPPT does not teach or suggest that the destination port *range* (or a port mask) is used to identify the destination IP telephone as required by these claims.

3. *Fangman Does Not Teach The Configuration Message (Claims 7, 10, 11, and 14)*

Fangman does not teach the configuration message of claims 7, 10, 11, and 14. Each claim includes a configuration message with three elements: 1) a primary address, 2) a port

mask, and 3) a routing identifier. As explained below, Fangman does not include either a port mask or a routing identifier in a configuration message.

With respect to claims 7 and 10, the Office Action points to three parts of Fangman discussing the DHCP service and information that an IP telephone may know. Office Action at 29-30 (citing Fangman at 3:26-29, 8:48-53, 11:64-67). As noted above, DHCP is defined by RFC 2131 and includes a options portion that is further defined by RFC 1533. *See generally* EX2003; EX2004. At a high level, in DHCP, a client device sends a DHCP discover message to obtain configuration parameters from a server EX2003 at 6, 13. The server responds with a DHCP offer message that includes the requested parameters. *Id.* at 13. A list of the contents of the DHCP service is included in Fangman's specification:

DHCP Service:

Standard features:

- Permanent lease with updates
- IP address
- Mask
- Default gateway
- Domain name
- Host name
- DNS server
- Validate address before assigning
- MAC based reservations
- Boot file name
- Boot file location

Options:

- Assign port ranges for host, reserved in NAPT table
- Assign soft switch address
- DHCP table synchronization with standby device

Additional DHCP Features:

- Support multiple scopes and super-scopes
- Support for MAC-based reservations
- Support leasing based on class identifier (ignore requests from non-phones)
- Issue leases based on client identifier (Ignore requests from non-phones)
- Support standard DHCP options:
 - Option 1 (Subnet Mask)
 - Option 3 (Router Option)

Option 4 (Time Server Option)
Option 13 (Boot File Size Option)
Option 17 (Root Path)
Option 50 (Requested IP Address)
Option 54 (Server Identifier)
Option 66 (TFTP Server Name)
Option 43 (Vendor Specific Information)
Vendor Specific Options:
Assign unique port ranges to DHCP clients
Assign primary/backup MGC IP address
Assign NAT address

Fangman at 10:44-11:15; *see also* EX2003 at 10, EX2004 (listing option fields).

Notably, the above contents of a DHCP offer do not include a port mask or a “routing identifier” or any equivalent. And, to the extent that DHCP includes a “mask,” it is a subnet mask defining a range of IP addresses. EX2003 at 45. Nor is it clear that anything in the DHCP request could meet the claimed “routing identifier.”

The portions of Fangman quoted in the Office Action highlight this. The first section explains that a DHCP offer message “may include DHCP lease information based on the validated identifier, such as the range of port numbers and information indicating operational software for the IP telephone.” Fangman at 3:26-29. Not only does this not mention masks, but the “identifier” refers to something to identify the IP telephone, such as a vendor ID, not a routing identifier. *Id.* at 3:10-13.

The second cite section of Fangman lists some of the standard features of DHCP, which as noted above, does not include a port mask or routing identifier:

Specifically, via DHCP the SG 170 may perform one or more of the following: assign the IP address and network mask, set the SG 170 as the default gateway, set the DNS domain to the appropriate value, set the SG 170 as the TFTP/BOOTP server, provide the path and file name for the boot image, and set the host name.

Id. at 8:48-53.

The last cited section of Fangman states that “The IP telephone 120 may also know the public IP address to which all its traffic is being NATed....” *Id.* at 11:64-67. But this does not suggest how the IP telephone may know the public IP address, nor does it suggest modifying the DHCP offer to include multiple IP addresses.

Further, Fangman does not teach the claimed primary address. These claims require that the primary address is “shared with a plurality of pieces of terminal equipment.” This primary address is the sent to the terminal in a configuration message. The Office Action contends that a “single public IP address” providing NAT support is the primary address. Office Action at 29, 31 (citing Fangman at 8:6-17).⁴ As noted above, the Office Action then points to the DHCP offer as the claimed message. *Id.* at 29-30. However, these sections from Fangman do not teach that the public address is included in a DHCP offer. Instead, this portion of Fangman suggests that a private address is transmitted in the DHCP offer. *See* Fangman at 8:48-53. Nothing cited in the Office Action with respect to these claims suggests that Fangman ever provides the IP telephone with the public address.⁵

Accordingly, Fangman does not describe the information contained in configuration message or data of independent claims 7, 10, 11, 14, and their dependents, and therefore, Fangman does not anticipate these claims.

⁴ With respect to claims 11 and 14, the Office Action does not specifically identify the primary address shared with a plurality of pieces of terminal equipment. Nonetheless, Fangman does not teach these claims for the same reasons.

⁵ With respect to Claim 14, the Office Action notes that the final element, which includes the primary address, is not a limitation because it does not recite any structural limitations. Patent Owner respectfully disagrees.

VII. CONCLUSION

For the foregoing reasons, Patent Owner respectfully requests that the Office find all claims patentable.

Dated: August 7, 2024

Respectfully submitted,
By: /Michael F. Heim/
Michael F. Heim (Reg. No. 32,702)
Christopher L. Limbacher (Reg. No. 76,551)
Attorneys for Patent Owner
Monarch Networking Solutions LLC

CERTIFICATE OF SERVICE

The undersigned certifies that a copy of the foregoing PATENT OWNER'S RESPONSE TO NON-FINAL OFFICE ACTION and IDS under 37 CFR 3.73 was served on Counsel for Requester by First Class Mail:

Counsel for Requester:
Theodore M. Foster
(Reg. No. 57,456)
HAYNES AND BOONE, LLP
2323 Victory Ave., Suite 700
Dallas, TX 75219
ipr.theo.foster@haynesboone.com

Dated: August 7, 2024

Respectfully submitted,
By: /Michael F. Heim/
Michael F. Heim (Reg. No. 32,702)
Christopher L. Limbacher (Reg. No. 76,551)
Attorneys for Patent Owner
Monarch Networking Solutions LLC

CERTIFICATE OF EFS-WEB TRANSMISSION

I hereby certify that this correspondence is being transmitted via the U.S. Patent and Trademark Office (USPTO) electronic filing system (EFS-Web) to the USPTO on August 7, 2024.

/Michael F. Heim/
Michael F. Heim (Reg. No. 32,702)
Christopher L. Limbacher (Reg. No. 76,551)
Attorneys for Patent Owner
Monarch Networking Solutions LLC

INTERNET PROTOCOL
DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

September 1981

prepared for
Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

by
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

TABLE OF CONTENTS

PREFACE	iii
1. INTRODUCTION	1
1.1 Motivation	1
1.2 Scope	1
1.3 Interfaces	1
1.4 Operation	2
2. OVERVIEW	5
2.1 Relation to Other Protocols	9
2.2 Model of Operation	5
2.3 Function Description	7
2.4 Gateways	9
3. SPECIFICATION	11
3.1 Internet Header Format	11
3.2 Discussion	23
3.3 Interfaces	31
APPENDIX A: Examples & Scenarios	34
APPENDIX B: Data Transmission Order	39
GLOSSARY	41
REFERENCES	45

PREFACE

This document specifies the DoD Standard Internet Protocol. This document is based on six earlier editions of the ARPA Internet Protocol Specification, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition revises aspects of addressing, error handling, option codes, and the security, precedence, compartments, and handling restriction features of the internet protocol.

Jon Postel

Editor

[Page iii]

RFC: 791
Replaces: RFC 760
IENs 128, 123, 111,
80, 54, 44, 41, 28, 26

INTERNET PROTOCOL
DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

1. INTRODUCTION

1.1. Motivation

The Internet Protocol is designed for use in interconnected systems of packet-switched computer communication networks. Such a system has been called a "catenet" [1]. The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. The internet protocol also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through "small packet" networks.

1.2. Scope

The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols. The internet protocol can capitalize on the services of its supporting networks to provide various types and qualities of service.

1.3. Interfaces

This protocol is called on by host-to-host protocols in an internet environment. This protocol calls on local network protocols to carry the internet datagram to the next gateway or destination host.

For example, a TCP module would call on the internet module to take a TCP segment (including the TCP header and user data) as the data portion of an internet datagram. The TCP module would provide the addresses and other parameters in the internet header to the internet module as arguments of the call. The internet module would then create an internet datagram and call on the local network interface to transmit the internet datagram.

In the ARPANET case, for example, the internet module would call on a

Internet Protocol
Introduction

local net module which would add the 1822 leader [2] to the internet datagram creating an ARPANET message to transmit to the IMP. The ARPANET address would be derived from the internet address by the local network interface and would be the address of some host in the ARPANET, that host might be a gateway to other networks.

1.4. Operation

The internet protocol implements two basic functions: addressing and fragmentation.

The internet modules use the addresses carried in the internet header to transmit internet datagrams toward their destinations. The selection of a path for transmission is called routing.

The internet modules use fields in the internet header to fragment and reassemble internet datagrams when necessary for transmission through "small packet" networks.

The model of operation is that an internet module resides in each host engaged in internet communication and in each gateway that interconnects networks. These modules share common rules for interpreting address fields and for fragmenting and assembling internet datagrams. In addition, these modules (especially in gateways) have procedures for making routing decisions and other functions.

The internet protocol treats each internet datagram as an independent entity unrelated to any other internet datagram. There are no connections or logical circuits (virtual or otherwise).

The internet protocol uses four key mechanisms in providing its service: Type of Service, Time to Live, Options, and Header Checksum.

The Type of Service is used to indicate the quality of the service desired. The type of service is an abstract or generalized set of parameters which characterize the service choices provided in the networks that make up the internet. This type of service indication is to be used by gateways to select the actual transmission parameters for a particular network, the network to be used for the next hop, or the next gateway when routing an internet datagram.

The Time to Live is an indication of an upper bound on the lifetime of an internet datagram. It is set by the sender of the datagram and reduced at the points along the route where it is processed. If the time to live reaches zero before the internet datagram reaches its destination, the internet datagram is destroyed. The time to live can be thought of as a self destruct time limit.

[Page 2]

The Options provide for control functions needed or useful in some situations but unnecessary for the most common communications. The options include provisions for timestamps, security, and special routing.

The Header Checksum provides a verification that the information used in processing internet datagram has been transmitted correctly. The data may contain errors. If the header checksum fails, the internet datagram is discarded at once by the entity which detects the error.

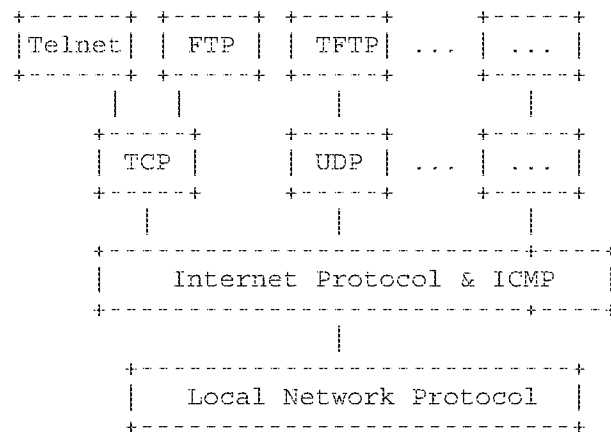
The internet protocol does not provide a reliable communication facility. There are no acknowledgments either end-to-end or hop-by-hop. There is no error control for data, only a header checksum. There are no retransmissions. There is no flow control.

Errors detected may be reported via the Internet Control Message Protocol (ICMP) [3] which is implemented in the internet protocol module.

2. OVERVIEW

2.1. Relation to Other Protocols

The following diagram illustrates the place of the internet protocol in the protocol hierarchy:



Protocol Relationships

Figure 1.

Internet protocol interfaces on one side to the higher level host-to-host protocols and on the other side to the local network protocol. In this context a "local network" may be a small network in a building or a large network such as the ARPANET.

2.2. Model of Operation

The model of operation for transmitting a datagram from one application program to another is illustrated by the following scenario:

We suppose that this transmission will involve one intermediate gateway.

The sending application program prepares its data and calls on its local internet module to send that data as a datagram and passes the destination address and other parameters as arguments of the call.

The internet module prepares a datagram header and attaches the data to it. The internet module determines a local network address for this internet address, in this case it is the address of a gateway.

[Page 5]

Internet Protocol Overview

It sends this datagram and the local network address to the local network interface.

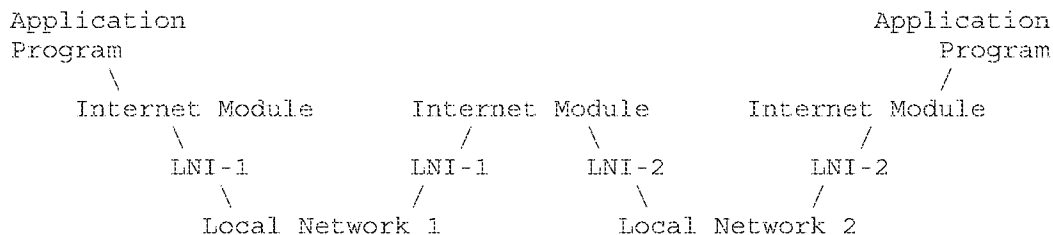
The local network interface creates a local network header, and attaches the datagram to it, then sends the result via the local network.

The datagram arrives at a gateway host wrapped in the local network header, the local network interface strips off this header, and turns the datagram over to the internet module. The internet module determines from the internet address that the datagram is to be forwarded to another host in a second network. The internet module determines a local net address for the destination host. It calls on the local network interface for that network to send the datagram.

This local network interface creates a local network header and attaches the datagram sending the result to the destination host.

At this destination host the datagram is stripped of the local net header by the local network interface and handed to the internet module.

The internet module determines that the datagram is for an application program in this host. It passes the data to the application program in response to a system call, passing the source address and other parameters as results of the call.



Transmission Path

Figure 2

2.3. Function Description

The function or purpose of Internet Protocol is to move datagrams through an interconnected set of networks. This is done by passing the datagrams from one internet module to another until the destination is reached. The internet modules reside in hosts and gateways in the internet system. The datagrams are routed from one internet module to another through individual networks based on the interpretation of an internet address. Thus, one important mechanism of the internet protocol is the internet address.

In the routing of messages from one internet module to another, datagrams may need to traverse a network whose maximum packet size is smaller than the size of the datagram. To overcome this difficulty, a fragmentation mechanism is provided in the internet protocol.

Addressing

A distinction is made between names, addresses, and routes [4]. A name indicates what we seek. An address indicates where it is. A route indicates how to get there. The internet protocol deals primarily with addresses. It is the task of higher level (i.e., host-to-host or application) protocols to make the mapping from names to addresses. The internet module maps internet addresses to local net addresses. It is the task of lower level (i.e., local net or gateways) procedures to make the mapping from local net addresses to routes.

Addresses are fixed length of four octets (32 bits). An address begins with a network number, followed by local address (called the "rest" field). There are three formats or classes of internet addresses: in class a, the high order bit is zero, the next 7 bits are the network, and the last 24 bits are the local address; in class b, the high order two bits are one-zero, the next 14 bits are the network and the last 16 bits are the local address; in class c, the high order three bits are one-one-zero, the next 21 bits are the network and the last 8 bits are the local address.

Care must be taken in mapping internet addresses to local net addresses; a single physical host must be able to act as if it were several distinct hosts to the extent of using several distinct internet addresses. Some hosts will also have several physical interfaces (multi-homing).

That is, provision must be made for a host to have several physical interfaces to the network with each having several logical internet addresses.

Internet Protocol
Overview

Examples of address mappings may be found in "Address Mappings" [5].

Fragmentation

Fragmentation of an internet datagram is necessary when it originates in a local net that allows a large packet size and must traverse a local net that limits packets to a smaller size to reach its destination.

An internet datagram can be marked "don't fragment." Any internet datagram so marked is not to be internet fragmented under any circumstances. If internet datagram marked don't fragment cannot be delivered to its destination without fragmenting it, it is to be discarded instead.

Fragmentation, transmission and reassembly across a local network which is invisible to the internet protocol module is called intranet fragmentation and may be used [6].

The internet fragmentation and reassembly procedure needs to be able to break a datagram into an almost arbitrary number of pieces that can be later reassembled. The receiver of the fragments uses the identification field to ensure that fragments of different datagrams are not mixed. The fragment offset field tells the receiver the position of a fragment in the original datagram. The fragment offset and length determine the portion of the original datagram covered by this fragment. The more-fragments flag indicates (by being reset) the last fragment. These fields provide sufficient information to reassemble datagrams.

The identification field is used to distinguish the fragments of one datagram from those of another. The originating protocol module of an internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system. The originating protocol module of a complete datagram sets the more-fragments flag to zero and the fragment offset to zero.

To fragment a long internet datagram, an internet protocol module (for example, in a gateway), creates two new internet datagrams and copies the contents of the internet header fields from the long datagram into both new internet headers. The data of the long datagram is divided into two portions on a 8 octet (64 bit) boundary (the second portion might not be an integral multiple of 8 octets, but the first must be). Call the number of 8 octet blocks in the first portion NFB (for Number of Fragment Blocks). The first portion of the data is placed in the first new internet datagram, and the total length field is set to the length of the first

datagram. The more-fragments flag is set to one. The second portion of the data is placed in the second new internet datagram, and the total length field is set to the length of the second datagram. The more-fragments flag carries the same value as the long datagram. The fragment offset field of the second new internet datagram is set to the value of that field in the long datagram plus NFB.

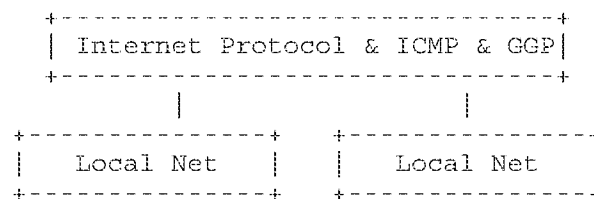
This procedure can be generalized for an n-way split, rather than the two-way split described.

To assemble the fragments of an internet datagram, an internet protocol module (for example at a destination host) combines internet datagrams that all have the same value for the four fields: identification, source, destination, and protocol. The combination is done by placing the data portion of each fragment in the relative position indicated by the fragment offset in that fragment's internet header. The first fragment will have the fragment offset zero, and the last fragment will have the more-fragments flag reset to zero.

2.4. Gateways

Gateways implement internet protocol to forward datagrams between networks. Gateways also implement the Gateway to Gateway Protocol (GGP) [7] to coordinate routing and other internet control information.

In a gateway the higher level protocols need not be implemented and the GGP functions are added to the IP module.



Gateway Protocols

Figure 3.

3. SPECIFICATION

3.1. Internet Header Format

A summary of the contents of the internet header follows:

0				1				2				3			
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5
Version				IHL				Type of Service				Total Length			
				Identification				Flags				Fragment Offset			
				Time to Live				Protocol				Header Checksum			
				Source Address											
				Destination Address											
				Options								Padding			

Example Internet Datagram Header

Figure 4.

Note that each tick mark represents one bit position.

Version: 4 bits

The Version field indicates the format of the internet header. This document describes version 4.

IHL: 4 bits

Internet Header Length is the length of the internet header in 32 bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5.

Type of Service: 8 bits

The Type of Service provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Several networks offer service precedence, which somehow treats high precedence traffic as more important than other traffic (generally by accepting only traffic above a certain precedence at time of high load). The major choice is a three way tradeoff between low-delay, high-reliability, and high-throughput.

Bits 0-2: Precedence.

Bit 3: 0 = Normal Delay, 1 = Low Delay.

Bits 4: 0 = Normal Throughput, 1 = High Throughput.

Bits 5: 0 = Normal Reliability, 1 = High Reliability.

Bit 6-7: Reserved for Future Use.

0	1	2	3	4	5	6	7
PRECEDENCE			D	T	R	0	0

Precedence

- 111 - Network Control
- 110 - Internetwork Control
- 101 - CRITIC/ECP
- 100 - Flash Override
- 011 - Flash
- 010 - Immediate
- 001 - Priority
- 000 - Routine

The use of the Delay, Throughput, and Reliability indications may increase the cost (in some sense) of the service. In many networks better performance for one of these parameters is coupled with worse performance on another. Except for very unusual cases at most two of these three indications should be set.

The type of service is used to specify the treatment of the datagram during its transmission through the internet system. Example mappings of the internet type of service to the actual service provided on networks such as AUTODIN II, ARPANET, SATNET, and PRNET is given in "Service Mappings" [8].

The Network Control precedence designation is intended to be used within a network only. The actual use and control of that designation is up to each network. The Internetwork Control designation is intended for use by gateway control originators only. If the actual use of these precedence designations is of concern to a particular network, it is the responsibility of that network to control the access to, and use of, those precedence designations.

Total Length: 16 bits

Total Length is the length of the datagram, measured in octets, including internet header and data. This field allows the length of a datagram to be up to 65,535 octets. Such long datagrams are impractical for most hosts and networks. All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole or in fragments). It is recommended that hosts only send datagrams larger than 576 octets if they have assurance that the destination is prepared to accept the larger datagrams.

The number 576 is selected to allow a reasonable sized data block to be transmitted in addition to the required header information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram. The maximal internet header is 60 octets, and a typical internet header is 20 octets, allowing a margin for headers of higher level protocols.

Identification: 16 bits

An identifying value assigned by the sender to aid in assembling the fragments of a datagram.

Flags: 3 bits

Various Control Flags.

Bit 0: reserved, must be zero

Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.

Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.

0	1	2
+	+	+
	D	M
	F	F
+	+	+

Fragment Offset: 13 bits

This field indicates where in the datagram this fragment belongs.

Internet Protocol
Specification

The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.

Time to Live: 8 bits

This field indicates the maximum time the datagram is allowed to remain in the internet system. If this field contains the value zero, then the datagram must be destroyed. This field is modified in internet header processing. The time is measured in units of seconds, but since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Protocol: 8 bits

This field indicates the next level protocol used in the data portion of the internet datagram. The values for various protocols are specified in "Assigned Numbers" [9].

Header Checksum: 16 bits

A checksum on the header only. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed.

The checksum algorithm is:

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

This is a simple to compute checksum and experimental evidence indicates it is adequate, but it is provisional and may be replaced by a CRC procedure, depending on further experience.

Source Address: 32 bits

The source address. See section 3.2.

Destination Address: 32 bits

The destination address. See section 3.2.

Options: variable

The options may appear or not in datagrams. They must be implemented by all IP modules (host and gateways). What is optional is their transmission in any particular datagram, not their implementation.

In some environments the security option may be required in all datagrams.

The option field is variable in length. There may be zero or more options. There are two cases for the format of an option:

Case 1: A single octet of option-type.

Case 2: An option-type octet, an option-length octet, and the actual option-data octets.

The option-length octet counts the option-type octet and the option-length octet as well as the option-data octets.

The option-type octet is viewed as having 3 fields:

1 bit copied flag,
2 bits option class,
5 bits option number.

The copied flag indicates that this option is copied into all fragments on fragmentation.

0 = not copied
1 = copied

The option classes are:

0 = control
1 = reserved for future use
2 = debugging and measurement
3 = reserved for future use

The following internet options are defined:

CLASS	NUMBER	LENGTH	DESCRIPTION
0	0	-	End of Option list. This option occupies only 1 octet; it has no length octet.
0	1	-	No Operation. This option occupies only 1 octet; it has no length octet.
0	2	11	Security. Used to carry Security, Compartmentation, User Group (TCC), and Handling Restriction Codes compatible with DOD requirements.
0	3	var.	Loose Source Routing. Used to route the internet datagram based on information supplied by the source.
0	9	var.	Strict Source Routing. Used to route the internet datagram based on information supplied by the source.
0	7	var.	Record Route. Used to trace the route an internet datagram takes.
0	8	4	Stream ID. Used to carry the stream identifier.
2	4	var.	Internet Timestamp.

Specific Option Definitions

End of Option List

```
+-----+
|00000000|
+-----+
  Type=0
```

This option indicates the end of the option list. This might not coincide with the end of the internet header according to the internet header length. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the internet header.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

No Operation

```

+-----+
|00000001|
+-----+
Type=1

```

This option may be used between options, for example, to align the beginning of a subsequent option on a 32 bit boundary.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

Security

This option provides a way for hosts to send security, compartmentation, handling restrictions, and TCC (closed user group) parameters. The format for this option is as follows:

```

+-----+-----+---//---+---//---+---//---+---//---+
|10000010|00001011|SSS SSS|CCC CCC|HHH HHH| TCC  |
+-----+-----+---//---+---//---+---//---+---//---+
Type=130 Length=11

```

Security (S field): 16 bits

Specifies one of 16 levels of security (eight of which are reserved for future use).

```

00000000 00000000 - Unclassified
11110001 00110101 - Confidential
01111000 10011010 - EFTO
10111100 01001101 - MMMM
01011110 00100110 - PROG
10101111 00010011 - Restricted
11010111 10001000 - Secret
01101011 11000101 - Top Secret
00110101 11100010 - (Reserved for future use)
10011010 11110001 - (Reserved for future use)
01001101 01111000 - (Reserved for future use)
00100100 10111101 - (Reserved for future use)
00010011 01011110 - (Reserved for future use)
10001001 10101111 - (Reserved for future use)
11000100 11010110 - (Reserved for future use)
11100010 01101011 - (Reserved for future use)

```

Internet Protocol
Specification

Compartments (C field): 16 bits

An all zero value is used when the information transmitted is not compartmented. Other values for the compartments field may be obtained from the Defense Intelligence Agency.

Handling Restrictions (H field): 16 bits

The values for the control and release markings are alphanumeric digraphs and are defined in the Defense Intelligence Agency Manual DIAM 65-19, "Standard Security Markings".

Transmission Control Code (TCC field): 24 bits

Provides a means to segregate traffic and define controlled communities of interest among subscribers. The TCC values are trigraphs, and are available from HQ DCA Code 530.

Must be copied on fragmentation. This option appears at most once in a datagram.

Loose Source and Record Route

```

+-----+-----+-----+-----+//-----+
|10000011| length | pointer|      route data      |
+-----+-----+-----+-----+//-----+
Type=131

```

The loose source and record route (LSRR) option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A route data is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the source route is empty (and the recorded route full) and the routing is to be based on the destination address field.

If the address in destination address field has been reached and the pointer is not greater than the length, the next address in the source route replaces the address in the destination address field, and the recorded route address replaces the source address just used, and pointer is increased by four.

The recorded route address is the internet module's own internet address as known in the environment into which this datagram is being forwarded.

This procedure of replacing the source route with the recorded route (though it is in the reverse of the order it must be in to be used as a source route) means the option (and the IP header as a whole) remains a constant length as the datagram progresses through the internet.

This option is a loose source route because the gateway or host IP is allowed to use any route of any number of other intermediate gateways to reach the next address in the route.

Must be copied on fragmentation. Appears at most once in a datagram.

Strict Source and Record Route

```
+-----+-----+-----+-----//-----+
|10001001| length | pointer|      route data      |
+-----+-----+-----+-----//-----+
Type=137
```

The strict source and record route (SSRR) option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A route data is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the source route is empty (and the

recorded route full) and the routing is to be based on the destination address field.

If the address in destination address field has been reached and the pointer is not greater than the length, the next address in the source route replaces the address in the destination address field, and the recorded route address replaces the source address just used, and pointer is increased by four.

The recorded route address is the internet module's own internet address as known in the environment into which this datagram is being forwarded.

This procedure of replacing the source route with the recorded route (though it is in the reverse of the order it must be in to be used as a source route) means the option (and the IP header as a whole) remains a constant length as the datagram progresses through the internet.

This option is a strict source route because the gateway or host IP must send the datagram directly to the next address in the source route through only the directly connected network indicated in the next address to reach the next gateway or host specified in the route.

Must be copied on fragmentation. Appears at most once in a datagram.

Record Route

```
+-----+-----+-----+-----+//-----+
|00000111| length | pointer|      route data      |
+-----+-----+-----+-----+//-----+
Type=7
```

The record route option provides a means to record the route of an internet datagram.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next area to store a route address. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A recorded route is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is

greater than the length, the recorded route data area is full. The originating host must compose this option with a large enough route data area to hold all the address expected. The size of the option does not change due to adding addresses. The initial contents of the route data area must be zero.

When an internet module routes a datagram it checks to see if the record route option is present. If it is, it inserts its own internet address as known in the environment into which this datagram is being forwarded into the recorded route beginning at the octet indicated by the pointer, and increments the pointer by four.

If the route data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the address into the recorded route. If there is some room but not enough room for a full address to be inserted, the original datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message may be sent to the source host [3].

Not copied on fragmentation, goes in first fragment only.
Appears at most once in a datagram.

Stream Identifier

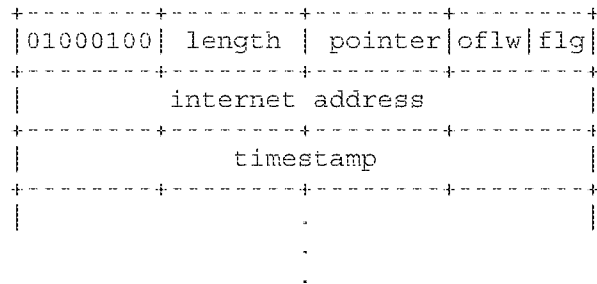
```
+-----+-----+-----+-----+
|10001000|00000010|   Stream ID   |
+-----+-----+-----+-----+
```

Type=136 Length=4

This option provides a way for the 16-bit SATNET stream identifier to be carried through networks that do not support the stream concept.

Must be copied on fragmentation. Appears at most once in a datagram.

Internet Timestamp



Type = 68

The Option Length is the number of octets in the option counting the type, length, pointer, and overflow/flag octets (maximum length 40).

The Pointer is the number of octets from the beginning of this option to the end of timestamps plus one (i.e., it points to the octet beginning the space for next timestamp). The smallest legal value is 5. The timestamp area is full when the pointer is greater than the length.

The Overflow (oflw) [4 bits] is the number of IP modules that cannot register timestamps due to lack of space.

The Flag (flg) [4 bits] values are

- 0 -- time stamps only, stored in consecutive 32-bit words,
- 1 -- each timestamp is preceded with internet address of the registering entity,
- 3 -- the internet address fields are prespecified. An IP module only registers its timestamp if it matches its own address with the next specified internet address.

The Timestamp is a right-justified, 32-bit timestamp in milliseconds since midnight UT. If the time is not available in milliseconds or cannot be provided with respect to midnight UT then any time may be inserted as a timestamp provided the high order bit of the timestamp field is set to one to indicate the use of a non-standard value.

The originating host must compose this option with a large enough timestamp data area to hold all the timestamp information expected. The size of the option does not change due to adding

timestamps. The initial contents of the timestamp data area must be zero or internet address/zero pairs.

If the timestamp data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the timestamp, but the overflow count is incremented by one.

If there is some room but not enough room for a full timestamp to be inserted, or the overflow count itself overflows, the original datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message may be sent to the source host [3].

The timestamp option is not copied upon fragmentation. It is carried in the first fragment. Appears at most once in a datagram.

Padding: variable

The internet header padding is used to ensure that the internet header ends on a 32 bit boundary. The padding is zero.

3.2. Discussion

The implementation of a protocol must be robust. Each implementation must expect to interoperate with others created by different individuals. While the goal of this specification is to be explicit about the protocol there is the possibility of differing interpretations. In general, an implementation must be conservative in its sending behavior, and liberal in its receiving behavior. That is, it must be careful to send well-formed datagrams, but must accept any datagram that it can interpret (e.g., not object to technical errors where the meaning is still clear).

The basic internet service is datagram oriented and provides for the fragmentation of datagrams at gateways, with reassembly taking place at the destination internet protocol module in the destination host. Of course, fragmentation and reassembly of datagrams within a network or by private agreement between the gateways of a network is also allowed since this is transparent to the internet protocols and the higher-level protocols. This transparent type of fragmentation and reassembly is termed "network-dependent" (or intranet) fragmentation and is not discussed further here.

Internet addresses distinguish sources and destinations to the host level and provide a protocol field as well. It is assumed that each protocol will provide for whatever multiplexing is necessary within a host.

Addressing

To provide for flexibility in assigning address to networks and allow for the large number of small to intermediate sized networks the interpretation of the address field is coded to specify a small number of networks with a large number of host, a moderate number of networks with a moderate number of hosts, and a large number of networks with a small number of hosts. In addition there is an escape code for extended addressing mode.

Address Formats:

High Order Bits	Format	Class
0	7 bits of net, 24 bits of host	a
10	14 bits of net, 16 bits of host	b
110	21 bits of net, 8 bits of host	c
111	escape to extended addressing mode	

A value of zero in the network field means this network. This is only used in certain ICMP messages. The extended addressing mode is undefined. Both of these features are reserved for future use.

The actual values assigned for network addresses is given in "Assigned Numbers" [9].

The local address, assigned by the local network, must allow for a single physical host to act as several distinct internet hosts. That is, there must be a mapping between internet host addresses and network/host interfaces that allows several internet addresses to correspond to one interface. It must also be allowed for a host to have several physical interfaces and to treat the datagrams from several of them as if they were all addressed to a single host.

Address mappings between internet addresses and addresses for ARPANET, SATNET, PRNET, and other networks are described in "Address Mappings" [5].

Fragmentation and Reassembly.

The internet identification field (ID) is used together with the source and destination address, and the protocol fields, to identify datagram fragments for reassembly.

The More Fragments flag bit (MF) is set if the datagram is not the last fragment. The Fragment Offset field identifies the fragment location, relative to the beginning of the original unfragmented datagram. Fragments are counted in units of 8 octets. The

fragmentation strategy is designed so that an unfragmented datagram has all zero fragmentation information (MF = 0, fragment offset = 0). If an internet datagram is fragmented, its data portion must be broken on 8 octet boundaries.

This format allows $2^{13} = 8192$ fragments of 8 octets each for a total of 65,536 octets. Note that this is consistent with the datagram total length field (of course, the header is counted in the total length and not in the fragments).

When fragmentation occurs, some options are copied, but others remain with the first fragment only.

Every internet module must be able to forward a datagram of 68 octets without further fragmentation. This is because an internet header may be up to 60 octets, and the minimum fragment is 8 octets.

Every internet destination must be able to receive a datagram of 576 octets either in one piece or in fragments to be reassembled.

The fields which may be affected by fragmentation include:

- (1) options field
- (2) more fragments flag
- (3) fragment offset
- (4) internet header length field
- (5) total length field
- (6) header checksum

If the Don't Fragment flag (DF) bit is set, then internet fragmentation of this datagram is NOT permitted, although it may be discarded. This can be used to prohibit fragmentation in cases where the receiving host does not have sufficient resources to reassemble internet fragments.

One example of use of the Don't Fragment feature is to down line load a small host. A small host could have a boot strap program that accepts a datagram stores it in memory and then executes it.

The fragmentation and reassembly procedures are most easily described by examples. The following procedures are example implementations.

General notation in the following pseudo programs: " $\leq$ " means "less than or equal", " $\neq$ " means "not equal", " $=$ " means "equal", " $\leftarrow$ " means "is set to". Also, " x to y " includes x and excludes y ; for example, " 4 to 7 " would include 4 , 5 , and 6 (but not 7).

An Example Fragmentation Procedure

The maximum sized datagram that can be transmitted through the next network is called the maximum transmission unit (MTU).

If the total length is less than or equal the maximum transmission unit then submit this datagram to the next step in datagram processing; otherwise cut the datagram into two fragments, the first fragment being the maximum size, and the second fragment being the rest of the datagram. The first fragment is submitted to the next step in datagram processing, while the second fragment is submitted to this procedure in case it is still too large.

Notation:

FO - Fragment Offset
IHL - Internet Header Length
DF - Don't Fragment flag
MF - More Fragments flag
TL - Total Length
OFO - Old Fragment Offset
OIHL - Old Internet Header Length
OMF - Old More Fragments flag
OTL - Old Total Length
NFB - Number of Fragment Blocks
MTU - Maximum Transmission Unit

Procedure:

IF TL ≤ MTU THEN Submit this datagram to the next step
in datagram processing ELSE IF DF = 1 THEN discard the
datagram ELSE

To produce the first fragment:

- (1) Copy the original internet header;
- (2) OIHL ← IHL; OTL ← TL; OFO ← FO; OMF ← MF;
- (3) NFB ← (MTU-IHL*4)/8;
- (4) Attach the first NFB*8 data octets;
- (5) Correct the header:
MF ← 1; TL ← (IHL*4)+(NFB*8);
Recompute Checksum;
- (6) Submit this fragment to the next step in
datagram processing;

To produce the second fragment:

- (7) Selectively copy the internet header (some options
are not copied, see option definitions);
- (8) Append the remaining data;
- (9) Correct the header:
IHL ← (((OIHL*4)-(length of options not copied))+3)/4;

```
TL <- OTL - NFB*8 - (OIHL-IHL)*4);  
FO <- OFO + NFB; MF <- OMF; Recompute Checksum;  
(10) Submit this fragment to the fragmentation test; DONE.
```

In the above procedure each fragment (except the last) was made the maximum allowable size. An alternative might produce less than the maximum size datagrams. For example, one could implement a fragmentation procedure that repeatedly divided large datagrams in half until the resulting fragments were less than the maximum transmission unit size.

An Example Reassembly Procedure

For each datagram the buffer identifier is computed as the concatenation of the source, destination, protocol, and identification fields. If this is a whole datagram (that is both the fragment offset and the more fragments fields are zero), then any reassembly resources associated with this buffer identifier are released and the datagram is forwarded to the next step in datagram processing.

If no other fragment with this buffer identifier is on hand then reassembly resources are allocated. The reassembly resources consist of a data buffer, a header buffer, a fragment block bit table, a total data length field, and a timer. The data from the fragment is placed in the data buffer according to its fragment offset and length, and bits are set in the fragment block bit table corresponding to the fragment blocks received.

If this is the first fragment (that is the fragment offset is zero) this header is placed in the header buffer. If this is the last fragment (that is the more fragments field is zero) the total data length is computed. If this fragment completes the datagram (tested by checking the bits set in the fragment block table), then the datagram is sent to the next step in datagram processing; otherwise the timer is set to the maximum of the current timer value and the value of the time to live field from this fragment; and the reassembly routine gives up control.

If the timer runs out, the all reassembly resources for this buffer identifier are released. The initial setting of the timer is a lower bound on the reassembly waiting time. This is because the waiting time will be increased if the Time to Live in the arriving fragment is greater than the current timer value but will not be decreased if it is less. The maximum this timer value could reach is the maximum time to live (approximately 4.25 minutes). The current recommendation for the initial timer setting is 15 seconds. This may be changed as experience with

this protocol accumulates. Note that the choice of this parameter value is related to the buffer capacity available and the data rate of the transmission medium; that is, data rate times timer value equals buffer size (e.g., 10Kb/s X 15s = 150Kb).

Notation:

FO - Fragment Offset
IHL - Internet Header Length
MF - More Fragments flag
TTL - Time To Live
NFB - Number of Fragment Blocks
TL - Total Length
TDL - Total Data Length
BUFID - Buffer Identifier
RCVBT - Fragment Received Bit Table
TLB - Timer Lower Bound

Procedure:

```
(1)  BUFID <- source|destination|protocol|identification;
(2)  IF FO = 0 AND MF = 0
(3)    THEN IF buffer with BUFID is allocated
(4)      THEN flush all reassembly for this BUFID;
(5)      Submit datagram to next step; DONE.
(6)  ELSE IF no buffer with BUFID is allocated
(7)    THEN allocate reassembly resources
           with BUFID;
           TIMER <- TLB; TDL <- 0;
(8)    put data from fragment into data buffer with
           BUFID from octet FO*8 to
           octet (TL-(IHL*4))+FO*8;
(9)    set RCVBT bits from FO
           to FO+((TL-(IHL*4)+7)/8);
(10)   IF MF = 0 THEN TDL <- TL-(IHL*4)+(FO*8)
(11)   IF FO = 0 THEN put header in header buffer
(12)   IF TDL # 0
(13)     AND all RCVBT bits from 0
           to (TDL+7)/8 are set
(14)     THEN TL <- TDL+(IHL*4)
(15)     Submit datagram to next step;
(16)     free all reassembly resources
           for this BUFID; DONE.
(17)   TIMER <- MAX(TIMER,TTL);
(18)   give up until next fragment or timer expires;
(19) timer expires: flush all reassembly with this BUFID; DONE.
```

In the case that two or more fragments contain the same data

either identically or through a partial overlap, this procedure will use the more recently arrived copy in the data buffer and datagram delivered.

Identification

The choice of the Identifier for a datagram is based on the need to provide a way to uniquely identify the fragments of a particular datagram. The protocol module assembling fragments judges fragments to belong to the same datagram if they have the same source, destination, protocol, and Identifier. Thus, the sender must choose the Identifier to be unique for this source, destination pair and protocol for the time the datagram (or any fragment of it) could be alive in the internet.

It seems then that a sending protocol module needs to keep a table of Identifiers, one entry for each destination it has communicated with in the last maximum packet lifetime for the internet.

However, since the Identifier field allows 65,536 different values, some host may be able to simply use unique identifiers independent of destination.

It is appropriate for some higher level protocols to choose the identifier. For example, TCP protocol modules may retransmit an identical TCP segment, and the probability for correct reception would be enhanced if the retransmission carried the same identifier as the original transmission since fragments of either datagram could be used to construct a correct TCP segment.

Type of Service

The type of service (TOS) is for internet service quality selection. The type of service is specified along the abstract parameters precedence, delay, throughput, and reliability. These abstract parameters are to be mapped into the actual service parameters of the particular networks the datagram traverses.

Precedence. An independent measure of the importance of this datagram.

Delay. Prompt delivery is important for datagrams with this indication.

Throughput. High data rate is important for datagrams with this indication.

Reliability. A higher level of effort to ensure delivery is important for datagrams with this indication.

For example, the ARPANET has a priority bit, and a choice between "standard" messages (type 0) and "uncontrolled" messages (type 3), (the choice between single packet and multipacket messages can also be considered a service parameter). The uncontrolled messages tend to be less reliably delivered and suffer less delay. Suppose an internet datagram is to be sent through the ARPANET. Let the internet type of service be given as:

Precedence:	5
Delay:	0
Throughput:	1
Reliability:	1

In this example, the mapping of these parameters to those available for the ARPANET would be to set the ARPANET priority bit on since the Internet precedence is in the upper half of its range, to select standard messages since the throughput and reliability requirements are indicated and delay is not. More details are given on service mappings in "Service Mappings" [8].

Time to Live

The time to live is set by the sender to the maximum time the datagram is allowed to be in the internet system. If the datagram is in the internet system longer than the time to live, then the datagram must be destroyed.

This field must be decreased at each point that the internet header is processed to reflect the time spent processing the datagram. Even if no local information is available on the time actually spent, the field must be decremented by 1. The time is measured in units of seconds (i.e. the value 1 means one second). Thus, the maximum time to live is 255 seconds or 4.25 minutes. Since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Some higher level reliable connection protocols are based on assumptions that old duplicate datagrams will not arrive after a certain time elapses. The TTL is a way for such protocols to have an assurance that their assumption is met.

Options

The options are optional in each datagram, but required in implementations. That is, the presence or absence of an option is the choice of the sender, but each internet module must be able to parse every option. There can be several options present in the option field.

The options might not end on a 32-bit boundary. The internet header must be filled out with octets of zeros. The first of these would be interpreted as the end-of-options option, and the remainder as internet header padding.

Every internet module must be able to act on every option. The Security Option is required if classified, restricted, or compartmented traffic is to be passed.

Checksum

The internet header checksum is recomputed if the internet header is changed. For example, a reduction of the time to live, additions or changes to internet options, or due to fragmentation. This checksum at the internet level is intended to protect the internet header fields from transmission errors.

There are some applications where a few data bit errors are acceptable while retransmission delays are not. If the internet protocol enforced data correctness such applications could not be supported.

Errors

Internet protocol errors may be reported via the ICMP messages [3].

3.3. Interfaces

The functional description of user interfaces to the IP is, at best, fictional, since every operating system will have different facilities. Consequently, we must warn readers that different IP implementations may have different user interfaces. However, all IPs must provide a certain minimum set of services to guarantee that all IP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all IP implementations.

Internet protocol interfaces on one side to the local network and on the other side to either a higher level protocol or an application program. In the following, the higher level protocol or application

Internet Protocol Specification

program (or even a gateway program) will be called the "user" since it is using the internet module. Since internet protocol is a datagram protocol, there is minimal memory or state maintained between datagram transmissions, and each call on the internet protocol module by the user supplies all information necessary for the IP to perform the service requested.

An Example Upper Level Interface

The following two example calls satisfy the requirements for the user to internet protocol module communication ("=>" means returns):

SEND (src, dst, prot, TOS, TTL, BufPTR, len, Id, DF, opt => result)

where:

```
src = source address
dst = destination address
prot = protocol
TOS = type of service
TTL = time to live
BufPTR = buffer pointer
len = length of buffer
Id = Identifier
DF = Don't Fragment
opt = option data
result = response
    OK = datagram sent ok
    Error = error in arguments or local network error
```

Note that the precedence is included in the TOS and the security/compartments is passed as an option.

RCV (BufPTR, prot, => result, src, dst, TOS, len, opt)

where:

```
BufPTR = buffer pointer
prot = protocol
result = response
    OK = datagram received ok
    Error = error in arguments
len = length of buffer
src = source address
dst = destination address
TOS = type of service
opt = option data
```

When the user sends a datagram, it executes the SEND call supplying all the arguments. The internet protocol module, on receiving this call, checks the arguments and prepares and sends the message. If the arguments are good and the datagram is accepted by the local network, the call returns successfully. If either the arguments are bad, or the datagram is not accepted by the local network, the call returns unsuccessfully. On unsuccessful returns, a reasonable report must be made as to the cause of the problem, but the details of such reports are up to individual implementations.

When a datagram arrives at the internet protocol module from the local network, either there is a pending RECV call from the user addressed or there is not. In the first case, the pending call is satisfied by passing the information from the datagram to the user. In the second case, the user addressed is notified of a pending datagram. If the user addressed does not exist, an ICMP error message is returned to the sender, and the data is discarded.

The notification of a user may be via a pseudo interrupt or similar mechanism, as appropriate in the particular operating system environment of the implementation.

A user's RECV call may then either be immediately satisfied by a pending datagram, or the call may be pending until a datagram arrives.

The source address is included in the send call in case the sending host has several addresses (multiple physical connections or logical addresses). The internet module must check to see that the source address is one of the legal address for this host.

An implementation may also allow or require a call to the internet module to indicate interest in or reserve exclusive use of a class of datagrams (e.g., all those with a certain value in the protocol field).

This section functionally characterizes a USER/IP interface. The notation used is similar to most procedure of function calls in high level languages, but this usage is not meant to rule out trap type service calls (e.g., SVCs, UUOs, EMTs), or any other form of interprocess communication.

Internet Protocol

APPENDIX A: Examples & Scenarios

Example 1:

This is an example of the minimal data carrying internet datagram:

```

      0               1               2               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|Ver= 4 |IHL= 5 |Type of Service|           Total Length = 21      |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Identification = 111       |Flg=0|   Fragment Offset = 0   |
+-----+-----+-----+-----+-----+-----+-----+-----+
|   Time = 123   | Protocol = 1 |           header checksum           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     source address                    |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     destination address                |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           data           |
+-----+-----+

```

Example Internet Datagram

Figure 5.

Note that each tick mark represents one bit position.

This is a internet datagram in version 4 of internet protocol; the internet header consists of five 32 bit words, and the total length of the datagram is 21 octets. This datagram is a complete datagram (not a fragment).

Example 2:

In this example, we show first a moderate size internet datagram (452 data octets), then two internet fragments that might result from the fragmentation of this datagram if the maximum sized transmission allowed were 280 octets.

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|Ver= 4 |IHL= 5 |Type of Service|          Total Length = 472      |
+-----+-----+-----+-----+-----+-----+-----+-----+
|  Identification = 111      |Flg=0|      Fragment Offset = 0    |
+-----+-----+-----+-----+-----+-----+-----+-----+
|  Time = 123   | Protocol = 6 |          header checksum      |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     source address              |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     destination address         |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     data                          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     data                          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     data                          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     data                          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     data                          |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Example Internet Datagram

Figure 6.

Internet Protocol

Now the first fragment that results from splitting the datagram after 256 data octets.

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|Ver= 4 |IHL= 5 |Type of Service|          Total Length = 276      |
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Identification = 111      |Flg=1|      Fragment Offset = 0 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Time = 119   | Protocol = 6 |          Header Checksum      |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                source address                      |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                destination address                 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                data                                  |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                data                                  |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                data                                  |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                data                                  |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Example Internet Fragment

Figure 7.

And the second fragment.

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|Ver= 4 |IHL= 5 |Type of Service|          Total Length = 216      |
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Identification = 111      |Flg=0|  Fragment Offset  = 32  |
+-----+-----+-----+-----+-----+-----+-----+-----+
|    Time = 119   | Protocol = 6  |          Header Checksum      |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     source address                |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     destination address            |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     data                            |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     data                            |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     data                            |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     data                            |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Example Internet Fragment

Figure 8.

Internet Protocol

Example 3:

Here, we show an example of a datagram containing options:

0								1								2								3											
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1				
Ver= 4								IHL= 8								Type of Service								Total Length = 576											
Identification = 111																Flg=0				Fragment Offset = 0															
Time = 123								Protocol = 6								Header Checksum																			
source address																																			
destination address																																			
Opt. Code = x								Opt. Len.= 3								option value								Opt. Code = x											
Opt. Len. = 4								option value																Opt. Code = 1											
Opt. Code = y								Opt. Len. = 3								option value								Opt. Code = 0											
data																																			
data																																			
data																																			

Example Internet Datagram

Figure 9.

GLOSSARY

- 1822 BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.
- ARPANET leader The control information on an ARPANET message at the host-IMP interface.
- ARPANET message The unit of transmission between a host and an IMP in the ARPANET. The maximum size is about 1012 octets (8096 bits).
- ARPANET packet A unit of transmission used internally in the ARPANET between IMPs. The maximum size is about 126 octets (1008 bits).
- Destination The destination address, an internet header field.
- DF The Don't Fragment bit carried in the flags field.
- Flags An internet header field carrying various control flags.
- Fragment Offset This internet header field indicates where in the internet datagram a fragment belongs.
- GGP Gateway to Gateway Protocol, the protocol used primarily between gateways to control routing and other gateway functions.
- header Control information at the beginning of a message, segment, datagram, packet or block of data.
- ICMP Internet Control Message Protocol, implemented in the internet module, the ICMP is used from gateways to hosts and between hosts to report errors and make routing suggestions.

Internet Protocol
Glossary

Identification

An internet header field carrying the identifying value assigned by the sender to aid in assembling the fragments of a datagram.

IHL

The internet header field Internet Header Length is the length of the internet header measured in 32 bit words.

IMP

The Interface Message Processor, the packet switch of the ARPANET.

Internet Address

A four octet (32 bit) source or destination address consisting of a Network field and a Local Address field.

internet datagram

The unit of data exchanged between a pair of internet modules (includes the internet header).

internet fragment

A portion of the data of an internet datagram with an internet header.

Local Address

The address of a host within a network. The actual mapping of an internet local address on to the host addresses in a network is quite general, allowing for many to one mappings.

MF

The More-Fragments Flag carried in the internet header flags field.

module

An implementation, usually in software, of a protocol or other procedure.

more-fragments flag

A flag indicating whether or not this internet datagram contains the end of an internet datagram, carried in the internet header Flags field.

NFB

The Number of Fragment Blocks in a the data portion of an internet fragment. That is, the length of a portion of data measured in 8 octet units.

[Page 42]

octet	An eight bit byte.
Options	The internet header Options field may contain several options, and each option may be several octets in length.
Padding	The internet header Padding field is used to ensure that the data begins on 32 bit word boundary. The padding is zero.
Protocol	In this document, the next higher level protocol identifier, an internet header field.
Rest	The local address portion of an Internet Address.
Source	The source address, an internet header field.
TCP	Transmission Control Protocol: A host-to-host protocol for reliable communication in internet environments.
TCP Segment	The unit of data exchanged between TCP modules (including the TCP header).
TFTP	Trivial File Transfer Protocol: A simple file transfer protocol built on UDP.
Time to Live	An internet header field which indicates the upper bound on how long this internet datagram may exist.
TOS	Type of Service
Total Length	The internet header field Total Length is the length of the datagram in octets including internet header and data.
TTL	Time to Live

Internet Protocol
Glossary

Type of Service

An internet header field which indicates the type (or quality) of service for this internet datagram.

UDP

User Datagram Protocol: A user level protocol for transaction oriented applications.

User

The user of the internet protocol. This may be a higher level protocol module, an application program, or a gateway program.

Version

The Version field indicates the format of the internet header.

REFERENCES

- [1] Cerf, V., "The Catenet Model for Internetworking," Information Processing Techniques Office, Defense Advanced Research Projects Agency, IEN 48, July 1978.
- [2] Bolt Beranek and Newman, "Specification for the Interconnection of a Host and an IMP," BBN Technical Report 1822, Revised May 1978.
- [3] Postel, J., "Internet Control Message Protocol - DARPA Internet Program Protocol Specification," RFC 792, USC/Information Sciences Institute, September 1981.
- [4] Shoch, J., "Inter-Network Naming, Addressing, and Routing," COMPCON, IEEE Computer Society, Fall 1978.
- [5] Postel, J., "Address Mappings," RFC 796, USC/Information Sciences Institute, September 1981.
- [6] Shoch, J., "Packet Fragmentation in Inter-Network Protocols," Computer Networks, v. 3, n. 1, February 1979.
- [7] Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Beranek and Newman, August 1979.
- [8] Postel, J., "Service Mappings," RFC 795, USC/Information Sciences Institute, September 1981.
- [9] Postel, J., "Assigned Numbers," RFC 790, USC/Information Sciences Institute, September 1981.

TRANSMISSION CONTROL PROTOCOL

DARPA INTERNET PROGRAM

PROTOCOL SPECIFICATION

September 1981

prepared for

Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

by

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

TABLE OF CONTENTS

PREFACE	iii
1. INTRODUCTION	1
1.1 Motivation	1
1.2 Scope	2
1.3 About This Document	2
1.4 Interfaces	3
1.5 Operation	3
2. PHILOSOPHY	7
2.1 Elements of the Internetwork System	7
2.2 Model of Operation	7
2.3 The Host Environment	8
2.4 Interfaces	9
2.5 Relation to Other Protocols	9
2.6 Reliable Communication	9
2.7 Connection Establishment and Clearing	10
2.8 Data Communication	12
2.9 Precedence and Security	13
2.10 Robustness Principle	13
3. FUNCTIONAL SPECIFICATION	15
3.1 Header Format	15
3.2 Terminology	19
3.3 Sequence Numbers	24
3.4 Establishing a connection	30
3.5 Closing a Connection	37
3.6 Precedence and Security	40
3.7 Data Communication	40
3.8 Interfaces	44
3.9 Event Processing	52
GLOSSARY	79
REFERENCES	85

PREFACE

This document describes the DoD Standard Transmission Control Protocol (TCP). There have been nine earlier editions of the ARPA TCP specification on which this standard is based, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition clarifies several details and removes the end-of-letter buffer-size adjustments, and redescibes the letter mechanism as a push function.

Jon Postel

Editor

[Page iii]

RFC: 793
Replaces: RFC 761
IENs: 129, 124, 112, 81,
55, 44, 40, 27, 21, 5

TRANSMISSION CONTROL PROTOCOL

DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION

1. INTRODUCTION

The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks.

This document describes the functions to be performed by the Transmission Control Protocol, the program that implements it, and its interface to programs or users that require its services.

1.1. Motivation

Computer communication systems are playing an increasingly important role in military, government, and civilian environments. This document focuses its attention primarily on military computer communication requirements, especially robustness in the presence of communication unreliability and availability in the presence of congestion, but many of these problems are found in the civilian and government sector as well.

As strategic and tactical computer communication networks are developed and deployed, it is essential to provide means of interconnecting them and to provide standard interprocess communication protocols which can support a broad range of applications. In anticipation of the need for such standards, the Deputy Undersecretary of Defense for Research and Engineering has declared the Transmission Control Protocol (TCP) described herein to be a basis for DoD-wide inter-process communication protocol standardization.

TCP is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications. The TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks. Very few assumptions are made as to the reliability of the communication protocols below the TCP layer. TCP assumes it can obtain a simple, potentially unreliable datagram service from the lower level protocols. In principle, the TCP should be able to operate above a wide spectrum of communication systems ranging from hard-wired connections to packet-switched or circuit-switched networks.

[Page 1]

Transmission Control Protocol

Introduction

TCP is based on concepts first described by Cerf and Kahn in [1]. The TCP fits into a layered protocol architecture just above a basic Internet Protocol [2] which provides a way for the TCP to send and receive variable-length segments of information enclosed in internet datagram "envelopes". The internet datagram provides a means for addressing source and destination TCPs in different networks. The internet protocol also deals with any fragmentation or reassembly of the TCP segments required to achieve transport and delivery through multiple networks and interconnecting gateways. The internet protocol also carries information on the precedence, security classification and compartmentation of the TCP segments, so this information can be communicated end-to-end across multiple networks.

Protocol Layering

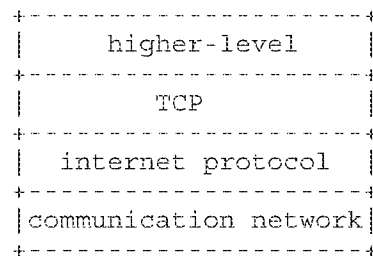


Figure 1

Much of this document is written in the context of TCP implementations which are co-resident with higher level protocols in the host computer. Some computer systems will be connected to networks via front-end computers which house the TCP and internet protocol layers, as well as network specific software. The TCP specification describes an interface to the higher level protocols which appears to be implementable even for the front-end case, as long as a suitable host-to-front end protocol is implemented.

1.2. Scope

The TCP is intended to provide a reliable process-to-process communication service in a multinet environment. The TCP is intended to be a host-to-host protocol in common use in multiple networks.

1.3. About this Document

This document represents a specification of the behavior required of any TCP implementation, both in its interactions with higher level protocols and in its interactions with other TCPs. The rest of this

[Page 2]

section offers a very brief view of the protocol interfaces and operation. Section 2 summarizes the philosophical basis for the TCP design. Section 3 offers both a detailed description of the actions required of TCP when various events occur (arrival of new segments, user calls, errors, etc.) and the details of the formats of TCP segments.

1.4. Interfaces

The TCP interfaces on one side to user or application processes and on the other side to a lower level protocol such as Internet Protocol.

The interface between an application process and the TCP is illustrated in reasonable detail. This interface consists of a set of calls much like the calls an operating system provides to an application process for manipulating files. For example, there are calls to open and close connections and to send and receive data on established connections. It is also expected that the TCP can asynchronously communicate with application programs. Although considerable freedom is permitted to TCP implementors to design interfaces which are appropriate to a particular operating system environment, a minimum functionality is required at the TCP/user interface for any valid implementation.

The interface between TCP and lower level protocol is essentially unspecified except that it is assumed there is a mechanism whereby the two levels can asynchronously pass information to each other. Typically, one expects the lower level protocol to specify this interface. TCP is designed to work in a very general environment of interconnected networks. The lower level protocol which is assumed throughout this document is the Internet Protocol [2].

1.5. Operation

As noted above, the primary purpose of the TCP is to provide reliable, securable logical circuit or connection service between pairs of processes. To provide this service on top of a less reliable internet communication system requires facilities in the following areas:

- Basic Data Transfer
- Reliability
- Flow Control
- Multiplexing
- Connections
- Precedence and Security

The basic operation of the TCP in each of these areas is described in the following paragraphs.

Transmission Control Protocol
Introduction

Basic Data Transfer:

The TCP is able to transfer a continuous stream of octets in each direction between its users by packaging some number of octets into segments for transmission through the internet system. In general, the TCPs decide when to block and forward data at their own convenience.

Sometimes users need to be sure that all the data they have submitted to the TCP has been transmitted. For this purpose a push function is defined. To assure that data submitted to a TCP is actually transmitted the sending user indicates that it should be pushed through to the receiving user. A push causes the TCPs to promptly forward and deliver data up to that point to the receiver. The exact push point might not be visible to the receiving user and the push function does not supply a record boundary marker.

Reliability:

The TCP must recover from data that is damaged, lost, duplicated, or delivered out of order by the internet communication system. This is achieved by assigning a sequence number to each octet transmitted, and requiring a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. At the receiver, the sequence numbers are used to correctly order segments that may be received out of order and to eliminate duplicates. Damage is handled by adding a checksum to each segment transmitted, checking it at the receiver, and discarding damaged segments.

As long as the TCPs continue to function properly and the internet system does not become completely partitioned, no transmission errors will affect the correct delivery of data. TCP recovers from internet communication system errors.

Flow Control:

TCP provides a means for the receiver to govern the amount of data sent by the sender. This is achieved by returning a "window" with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. The window indicates an allowed number of octets that the sender may transmit before receiving further permission.

Multiplexing:

To allow for many processes within a single Host to use TCP communication facilities simultaneously, the TCP provides a set of addresses or ports within each host. Concatenated with the network and host addresses from the internet communication layer, this forms a socket. A pair of sockets uniquely identifies each connection. That is, a socket may be simultaneously used in multiple connections.

The binding of ports to processes is handled independently by each Host. However, it proves useful to attach frequently used processes (e.g., a "logger" or timesharing service) to fixed sockets which are made known to the public. These services can then be accessed through the known addresses. Establishing and learning the port addresses of other processes may involve more dynamic mechanisms.

Connections:

The reliability and flow control mechanisms described above require that TCPs initialize and maintain certain status information for each data stream. The combination of this information, including sockets, sequence numbers, and window sizes, is called a connection. Each connection is uniquely specified by a pair of sockets identifying its two sides.

When two processes wish to communicate, their TCP's must first establish a connection (initialize the status information on each side). When their communication is complete, the connection is terminated or closed to free the resources for other uses.

Since connections must be established between unreliable hosts and over the unreliable internet communication system, a handshake mechanism with clock-based sequence numbers is used to avoid erroneous initialization of connections.

Precedence and Security:

The users of TCP may indicate the security and precedence of their communication. Provision is made for default values to be used when these features are not needed.

2. PHILOSOPHY

2.1. Elements of the Internetwork System

The internetwork environment consists of hosts connected to networks which are in turn interconnected via gateways. It is assumed here that the networks may be either local networks (e.g., the ETHERNET) or large networks (e.g., the ARPANET), but in any case are based on packet switching technology. The active agents that produce and consume messages are processes. Various levels of protocols in the networks, the gateways, and the hosts support an interprocess communication system that provides two-way data flow on logical connections between process ports.

The term packet is used generically here to mean the data of one transaction between a host and its network. The format of data blocks exchanged within the a network will generally not be of concern to us.

Hosts are computers attached to a network, and from the communication network's point of view, are the sources and destinations of packets. Processes are viewed as the active elements in host computers (in accordance with the fairly common definition of a process as a program in execution). Even terminals and files or other I/O devices are viewed as communicating with each other through the use of processes. Thus, all communication is viewed as inter-process communication.

Since a process may need to distinguish among several communication streams between itself and another process (or processes), we imagine that each process may have a number of ports through which it communicates with the ports of other processes.

2.2. Model of Operation

Processes transmit data by calling on the TCP and passing buffers of data as arguments. The TCP packages the data from these buffers into segments and calls on the internet module to transmit each segment to the destination TCP. The receiving TCP places the data from a segment into the receiving user's buffer and notifies the receiving user. The TCPs include control information in the segments which they use to ensure reliable ordered data transmission.

The model of internet communication is that there is an internet protocol module associated with each TCP which provides an interface to the local network. This internet module packages TCP segments inside internet datagrams and routes these datagrams to a destination internet module or intermediate gateway. To transmit the datagram through the local network, it is embedded in a local network packet.

The packet switches may perform further packaging, fragmentation, or

Transmission Control Protocol Philosophy

other operations to achieve the delivery of the local packet to the destination internet module.

At a gateway between networks, the internet datagram is "unwrapped" from its local packet and examined to determine through which network the internet datagram should travel next. The internet datagram is then "wrapped" in a local packet suitable to the next network and routed to the next gateway, or to the final destination.

A gateway is permitted to break up an internet datagram into smaller internet datagram fragments if this is necessary for transmission through the next network. To do this, the gateway produces a set of internet datagrams; each carrying a fragment. Fragments may be further broken into smaller fragments at subsequent gateways. The internet datagram fragment format is designed so that the destination internet module can reassemble fragments into internet datagrams.

A destination internet module unwraps the segment from the datagram (after reassembling the datagram, if necessary) and passes it to the destination TCP.

This simple model of the operation glosses over many details. One important feature is the type of service. This provides information to the gateway (or internet module) to guide it in selecting the service parameters to be used in traversing the next network. Included in the type of service information is the precedence of the datagram. Datagrams may also carry security information to permit host and gateways that operate in multilevel secure environments to properly segregate datagrams for security considerations.

2.3. The Host Environment

The TCP is assumed to be a module in an operating system. The users access the TCP much like they would access the file system. The TCP may call on other operating system functions, for example, to manage data structures. The actual interface to the network is assumed to be controlled by a device driver module. The TCP does not call on the network device driver directly, but rather calls on the internet datagram protocol module which may in turn call on the device driver.

The mechanisms of TCP do not preclude implementation of the TCP in a front-end processor. However, in such an implementation, a host-to-front-end protocol must provide the functionality to support the type of TCP-user interface described in this document.

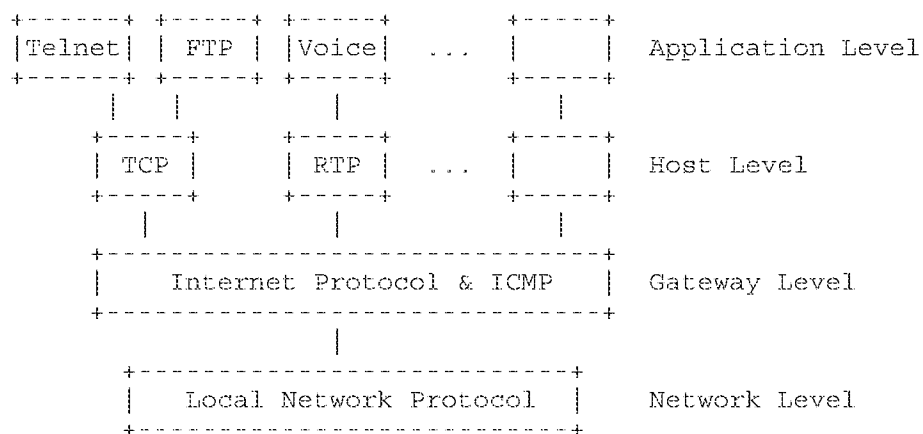
2.4. Interfaces

The TCP/user interface provides for calls made by the user on the TCP to OPEN or CLOSE a connection, to SEND or RECEIVE data, or to obtain STATUS about a connection. These calls are like other calls from user programs on the operating system, for example, the calls to open, read from, and close a file.

The TCP/internet interface provides calls to send and receive datagrams addressed to TCP modules in hosts anywhere in the internet system. These calls have parameters for passing the address, type of service, precedence, security, and other control information.

2.5. Relation to Other Protocols

The following diagram illustrates the place of the TCP in the protocol hierarchy:



Protocol Relationships

Figure 2.

It is expected that the TCP will be able to support higher level protocols efficiently. It should be easy to interface higher level protocols like the ARPANET Telnet or AUTODIN II THP to the TCP.

2.6. Reliable Communication

A stream of data sent on a TCP connection is delivered reliably and in order at the destination.

Transmission Control Protocol Philosophy

Transmission is made reliable via the use of sequence numbers and acknowledgments. Conceptually, each octet of data is assigned a sequence number. The sequence number of the first octet of data in a segment is transmitted with that segment and is called the segment sequence number. Segments also carry an acknowledgment number which is the sequence number of the next expected data octet of transmissions in the reverse direction. When the TCP transmits a segment containing data, it puts a copy on a retransmission queue and starts a timer; when the acknowledgment for that data is received, the segment is deleted from the queue. If the acknowledgment is not received before the timer runs out, the segment is retransmitted.

An acknowledgment by TCP does not guarantee that the data has been delivered to the end user, but only that the receiving TCP has taken the responsibility to do so.

To govern the flow of data between TCPs, a flow control mechanism is employed. The receiving TCP reports a "window" to the sending TCP. This window specifies the number of octets, starting with the acknowledgment number, that the receiving TCP is currently prepared to receive.

2.7. Connection Establishment and Clearing

To identify the separate data streams that a TCP may handle, the TCP provides a port identifier. Since port identifiers are selected independently by each TCP they might not be unique. To provide for unique addresses within each TCP, we concatenate an internet address identifying the TCP with a port identifier to create a socket which will be unique throughout all networks connected together.

A connection is fully specified by the pair of sockets at the ends. A local socket may participate in many connections to different foreign sockets. A connection can be used to carry data in both directions, that is, it is "full duplex".

TCPs are free to associate ports with processes however they choose. However, several basic concepts are necessary in any implementation. There must be well-known sockets which the TCP associates only with the "appropriate" processes by some means. We envision that processes may "own" ports, and that processes can initiate connections only on the ports they own. (Means for implementing ownership is a local issue, but we envision a Request Port user command, or a method of uniquely allocating a group of ports to a given process, e.g., by associating the high order bits of a port name with a given process.)

A connection is specified in the OPEN call by the local port and foreign socket arguments. In return, the TCP supplies a (short) local

connection name by which the user refers to the connection in subsequent calls. There are several things that must be remembered about a connection. To store this information we imagine that there is a data structure called a Transmission Control Block (TCB). One implementation strategy would have the local connection name be a pointer to the TCB for this connection. The OPEN call also specifies whether the connection establishment is to be actively pursued, or to be passively waited for.

A passive OPEN request means that the process wants to accept incoming connection requests rather than attempting to initiate a connection. Often the process requesting a passive OPEN will accept a connection request from any caller. In this case a foreign socket of all zeros is used to denote an unspecified socket. Unspecified foreign sockets are allowed only on passive OPENs.

A service process that wished to provide services for unknown other processes would issue a passive OPEN request with an unspecified foreign socket. Then a connection could be made with any process that requested a connection to this local socket. It would help if this local socket were known to be associated with this service.

Well-known sockets are a convenient mechanism for a priori associating a socket address with a standard service. For instance, the "Telnet-Server" process is permanently assigned to a particular socket, and other sockets are reserved for File Transfer, Remote Job Entry, Text Generator, Echoer, and Sink processes (the last three being for test purposes). A socket address might be reserved for access to a "Look-Up" service which would return the specific socket at which a newly created service would be provided. The concept of a well-known socket is part of the TCP specification, but the assignment of sockets to services is outside this specification. (See [4].)

Processes can issue passive OPENs and wait for matching active OPENs from other processes and be informed by the TCP when connections have been established. Two processes which issue active OPENs to each other at the same time will be correctly connected. This flexibility is critical for the support of distributed computing in which components act asynchronously with respect to each other.

There are two principal cases for matching the sockets in the local passive OPENs and an foreign active OPENs. In the first case, the local passive OPENs has fully specified the foreign socket. In this case, the match must be exact. In the second case, the local passive OPENs has left the foreign socket unspecified. In this case, any foreign socket is acceptable as long as the local sockets match. Other possibilities include partially restricted matches.

Transmission Control Protocol Philosophy

If there are several pending passive OPENs (recorded in TCBs) with the same local socket, an foreign active OPEN will be matched to a TCB with the specific foreign socket in the foreign active OPEN, if such a TCB exists, before selecting a TCB with an unspecified foreign socket.

The procedures to establish connections utilize the synchronize (SYN) control flag and involves an exchange of three messages. This exchange has been termed a three-way hand shake [3].

A connection is initiated by the rendezvous of an arriving segment containing a SYN and a waiting TCB entry each created by a user OPEN command. The matching of local and foreign sockets determines when a connection has been initiated. The connection becomes "established" when sequence numbers have been synchronized in both directions.

The clearing of a connection also involves the exchange of segments, in this case carrying the FIN control flag.

2.8. Data Communication

The data that flows on a connection may be thought of as a stream of octets. The sending user indicates in each SEND call whether the data in that call (and any preceeding calls) should be immediately pushed through to the receiving user by the setting of the PUSH flag.

A sending TCP is allowed to collect data from the sending user and to send that data in segments at its own convenience, until the push function is signaled, then it must send all unsent data. When a receiving TCP sees the PUSH flag, it must not wait for more data from the sending TCP before passing the data to the receiving process.

There is no necessary relationship between push functions and segment boundaries. The data in any particular segment may be the result of a single SEND call, in whole or part, or of multiple SEND calls.

The purpose of push function and the PUSH flag is to push data through from the sending user to the receiving user. It does not provide a record service.

There is a coupling between the push function and the use of buffers of data that cross the TCP/user interface. Each time a PUSH flag is associated with data placed into the receiving user's buffer, the buffer is returned to the user for processing even if the buffer is not filled. If data arrives that fills the user's buffer before a PUSH is seen, the data is passed to the user in buffer size units.

TCP also provides a means to communicate to the receiver of data that at some point further along in the data stream than the receiver is

currently reading there is urgent data. TCP does not attempt to define what the user specifically does upon being notified of pending urgent data, but the general notion is that the receiving process will take action to process the urgent data quickly.

2.9. Precedence and Security

The TCP makes use of the internet protocol type of service field and security option to provide precedence and security on a per connection basis to TCP users. Not all TCP modules will necessarily function in a multilevel secure environment; some may be limited to unclassified use only, and others may operate at only one security level and compartment. Consequently, some TCP implementations and services to users may be limited to a subset of the multilevel secure case.

TCP modules which operate in a multilevel secure environment must properly mark outgoing segments with the security, compartment, and precedence. Such TCP modules must also provide to their users or higher level protocols such as Telnet or THP an interface to allow them to specify the desired security level, compartment, and precedence of connections.

2.10. Robustness Principle

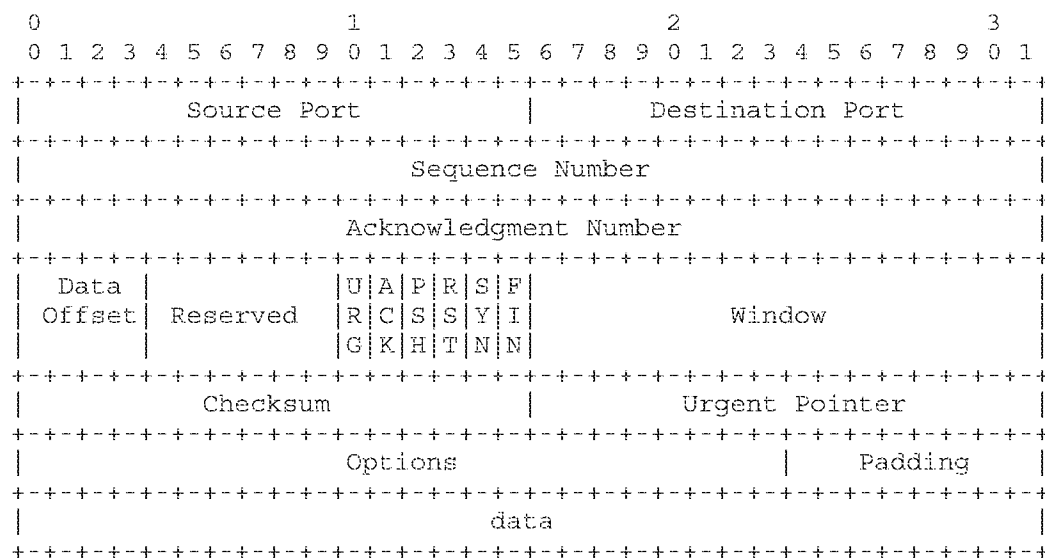
TCP implementations will follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others.

3. FUNCTIONAL SPECIFICATION

3.1. Header Format

TCP segments are sent as internet datagrams. The Internet Protocol header carries several information fields, including the source and destination host addresses [2]. A TCP header follows the internet header, supplying information specific to the TCP protocol. This division allows for the existence of host level protocols other than TCP.

TCP Header Format



TCP Header Format

Note that one tick mark represents one bit position.

Figure 3.

Source Port: 16 bits

The source port number.

Destination Port: 16 bits

The destination port number.

Transmission Control Protocol
Functional Specification

Sequence Number: 32 bits

The sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

Acknowledgment Number: 32 bits

If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

Data Offset: 4 bits

The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.

Reserved: 6 bits

Reserved for future use. Must be zero.

Control Bits: 6 bits (from left to right):

URG: Urgent Pointer field significant
ACK: Acknowledgment field significant
PSH: Push Function
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: No more data from sender

Window: 16 bits

The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.

Checksum: 16 bits

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.

The checksum also covers a 96 bit pseudo header conceptually

prefixed to the TCP header. This pseudo header contains the Source Address, the Destination Address, the Protocol, and TCP length. This gives the TCP protection against misrouted segments. This information is carried in the Internet Protocol and is transferred across the TCP/Network interface in the arguments or results of calls by the TCP on the IP.

Source Address		
Destination Address		
zero	PTCL	TCP Length

The TCP Length is the TCP header length plus the data length in octets (this is not an explicitly transmitted quantity, but is computed), and it does not count the 12 octets of the pseudo header.

Urgent Pointer: 16 bits

This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only be interpreted in segments with the URG control bit set.

Options: variable

Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin on any octet boundary. There are two cases for the format of an option:

Case 1: A single octet of option-kind.

Case 2: An octet of option-kind, an octet of option-length, and the actual option-data octets.

The option-length counts the two octets of option-kind and option-length as well as the option-data octets.

Note that the list of options may be shorter than the data offset field might imply. The content of the header beyond the End-of-Option option must be header padding (i.e., zero).

A TCP must implement all options.

Currently defined options include (kind indicated in octal):

Kind	Length	Meaning
----	-----	-----
0	-	End of option list.
1	-	No-Operation.
2	4	Maximum Segment Size.

Specific Option Definitions

End of Option List

```
+-----+
|00000000|
+-----+
Kind=0
```

This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

No-Operation

```
+-----+
|00000001|
+-----+
Kind=1
```

This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary.

Maximum Segment Size

```
+-----+-----+-----+-----+
|00000010|00000100|  max seg size  |
+-----+-----+-----+-----+
Kind=2   Length=4
```

Maximum Segment Size Option Data: 16 bits

If this option is present, then it communicates the maximum receive segment size at the TCP which sends this segment. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed.

Padding: variable

The TCP header padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary. The padding is composed of zeros.

3.2. Terminology

Before we can discuss very much about the operation of the TCP we need to introduce some detailed terminology. The maintenance of a TCP connection requires the remembering of several variables. We conceive of these variables being stored in a connection record called a Transmission Control Block or TCB. Among the variables stored in the TCB are the local and remote socket numbers, the security and precedence of the connection, pointers to the user's send and receive buffers, pointers to the retransmit queue and to the current segment. In addition several variables relating to the send and receive sequence numbers are stored in the TCB.

Send Sequence Variables

SND.UNA - send unacknowledged
SND.NXT - send next
SND.WND - send window
SND.UP - send urgent pointer
SND.WL1 - segment sequence number used for last window update
SND.WL2 - segment acknowledgment number used for last window update
ISS - initial send sequence number

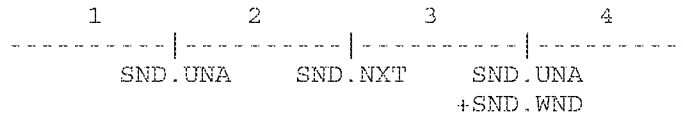
Receive Sequence Variables

RCV.NXT - receive next
RCV.WND - receive window
RCV.UP - receive urgent pointer
IRS - initial receive sequence number

Transmission Control Protocol Functional Specification

The following diagrams may help to relate some of these variables to the sequence space.

Send Sequence Space



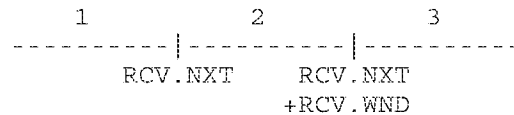
- 1 - old sequence numbers which have been acknowledged
- 2 - sequence numbers of unacknowledged data
- 3 - sequence numbers allowed for new data transmission
- 4 - future sequence numbers which are not yet allowed

Send Sequence Space

Figure 4.

The send window is the portion of the sequence space labeled 3 in figure 4.

Receive Sequence Space



- 1 - old sequence numbers which have been acknowledged
- 2 - sequence numbers allowed for new reception
- 3 - future sequence numbers which are not yet allowed

Receive Sequence Space

Figure 5.

The receive window is the portion of the sequence space labeled 2 in figure 5.

There are also some variables used frequently in the discussion that take their values from the fields of the current segment.

Current Segment Variables

SEG.SEQ - segment sequence number
SEG.ACK - segment acknowledgment number
SEG.LEN - segment length
SEG.WND - segment window
SEG.UP - segment urgent pointer
SEG.PRC - segment precedence value

A connection progresses through a series of states during its lifetime. The states are: LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, and the fictional state CLOSED. CLOSED is fictional because it represents the state when there is no TCB, and therefore, no connection. Briefly the meanings of the states are:

LISTEN - represents waiting for a connection request from any remote TCP and port.

SYN-SENT - represents waiting for a matching connection request after having sent a connection request.

SYN-RECEIVED - represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.

ESTABLISHED - represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.

FIN-WAIT-1 - represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent.

FIN-WAIT-2 - represents waiting for a connection termination request from the remote TCP.

CLOSE-WAIT - represents waiting for a connection termination request from the local user.

CLOSING - represents waiting for a connection termination request acknowledgment from the remote TCP.

LAST-ACK - represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request).

Transmission Control Protocol
Functional Specification

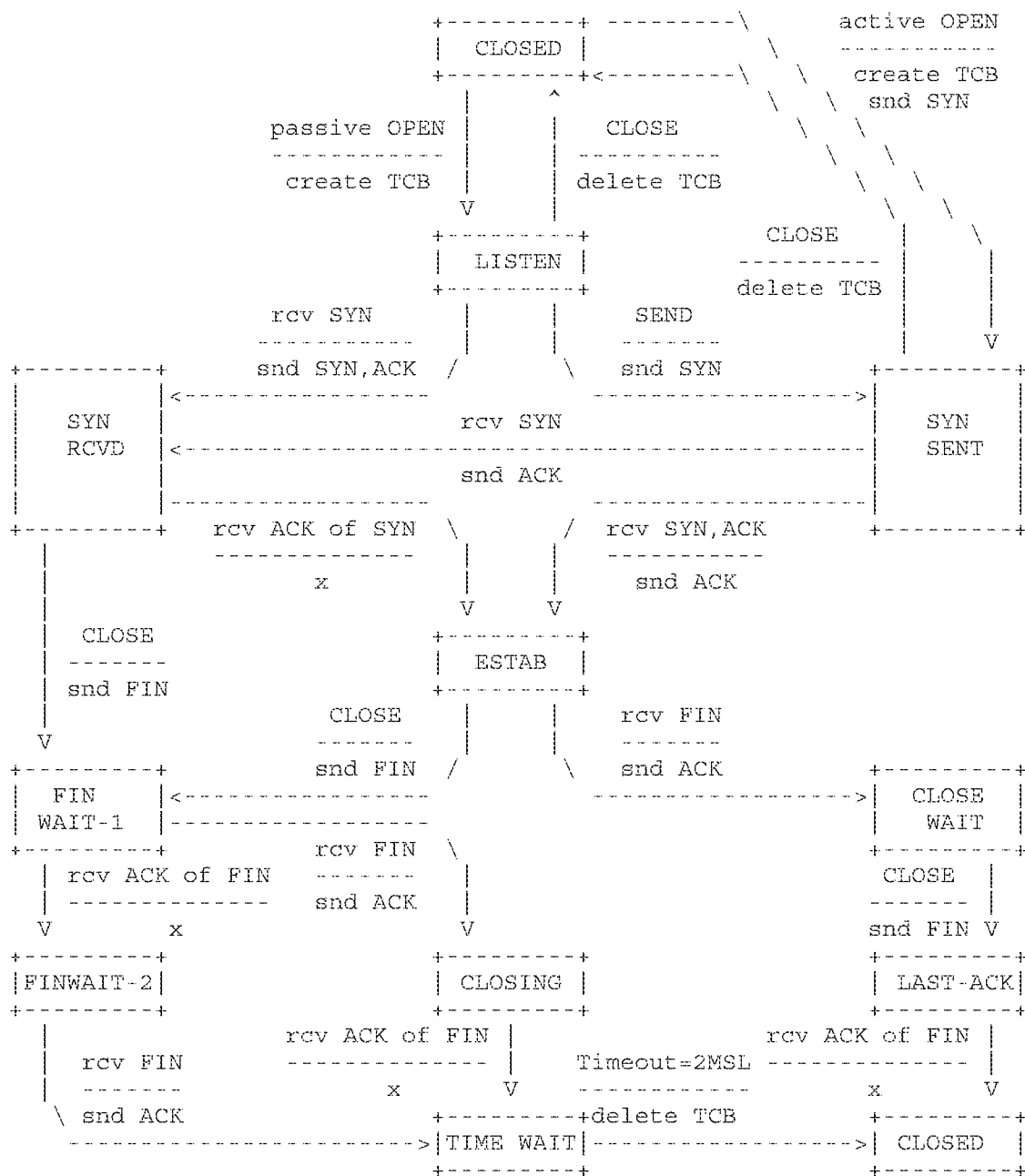
TIME-WAIT - represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request.

CLOSED - represents no connection state at all.

A TCP connection progresses from one state to another in response to events. The events are the user calls, OPEN, SEND, RECEIVE, CLOSE, ABORT, and STATUS; the incoming segments, particularly those containing the SYN, ACK, RST and FIN flags; and timeouts.

The state diagram in figure 6 illustrates only state changes, together with the causing events and resulting actions, but addresses neither error conditions nor actions which are not connected with state changes. In a later section, more detail is offered with respect to the reaction of the TCP to events.

NOTE BENE: this diagram is only a summary and must not be taken as the total specification.

TCP Connection State Diagram
Figure 6.

Transmission Control Protocol
Functional Specification

3.3. Sequence Numbers

A fundamental notion in the design is that every octet of data sent over a TCP connection has a sequence number. Since every octet is sequenced, each of them can be acknowledged. The acknowledgment mechanism employed is cumulative so that an acknowledgment of sequence number X indicates that all octets up to but not including X have been received. This mechanism allows for straight-forward duplicate detection in the presence of retransmission. Numbering of octets within a segment is that the first data octet immediately following the header is the lowest numbered, and the following octets are numbered consecutively.

It is essential to remember that the actual sequence number space is finite, though very large. This space ranges from 0 to $2^{32} - 1$. Since the space is finite, all arithmetic dealing with sequence numbers must be performed modulo 2^{32} . This unsigned arithmetic preserves the relationship of sequence numbers as they cycle from $2^{32} - 1$ to 0 again. There are some subtleties to computer modulo arithmetic, so great care should be taken in programming the comparison of such values. The symbol " $\leq$ " means "less than or equal" (modulo 2^{32}).

The typical kinds of sequence number comparisons which the TCP must perform include:

- (a) Determining that an acknowledgment refers to some sequence number sent but not yet acknowledged.
- (b) Determining that all sequence numbers occupied by a segment have been acknowledged (e.g., to remove the segment from a retransmission queue).
- (c) Determining that an incoming segment contains sequence numbers which are expected (i.e., that the segment "overlaps" the receive window).

In response to sending data the TCP will receive acknowledgments. The following comparisons are needed to process the acknowledgments.

SND.UNA = oldest unacknowledged sequence number

SND.NXT = next sequence number to be sent

SEG.ACK = acknowledgment from the receiving TCP (next sequence number expected by the receiving TCP)

SEG.SEQ = first sequence number of a segment

SEG.LEN = the number of octets occupied by the data in the segment (counting SYN and FIN)

SEG.SEQ+SEG.LEN-1 = last sequence number of a segment

A new acknowledgment (called an "acceptable ack"), is one for which the inequality below holds:

$$\text{SND.UNA} < \text{SEG.ACK} \leq \text{SND.NXT}$$

A segment on the retransmission queue is fully acknowledged if the sum of its sequence number and length is less or equal than the acknowledgment value in the incoming segment.

When data is received the following comparisons are needed:

RCV.NXT = next sequence number expected on an incoming segments, and is the left or lower edge of the receive window

RCV.NXT+RCV.WND-1 = last sequence number expected on an incoming segment, and is the right or upper edge of the receive window

SEG.SEQ = first sequence number occupied by the incoming segment

SEG.SEQ+SEG.LEN-1 = last sequence number occupied by the incoming segment

A segment is judged to occupy a portion of valid receive sequence space if

$$\text{RCV.NXT} \leq \text{SEG.SEQ} < \text{RCV.NXT} + \text{RCV.WND}$$

or

$$\text{RCV.NXT} \leq \text{SEG.SEQ} + \text{SEG.LEN} - 1 < \text{RCV.NXT} + \text{RCV.WND}$$

Transmission Control Protocol
Functional Specification

The first part of this test checks to see if the beginning of the segment falls in the window, the second part of the test checks to see if the end of the segment falls in the window; if the segment passes either part of the test it contains data in the window.

Actually, it is a little more complicated than this. Due to zero windows and zero length segments, we have four cases for the acceptability of an incoming segment:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT ≤ SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT ≤ SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT ≤ SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

Note that when the receive window is zero no segments should be acceptable except ACK segments. Thus, it is possible for a TCP to maintain a zero receive window while transmitting data and receiving ACKs. However, even when the receive window is zero, a TCP must process the RST and URG fields of all incoming segments.

We have taken advantage of the numbering scheme to protect certain control information as well. This is achieved by implicitly including some control flags in the sequence space so they can be retransmitted and acknowledged without confusion (i.e., one and only one copy of the control will be acted upon). Control information is not physically carried in the segment data space. Consequently, we must adopt rules for implicitly assigning sequence numbers to control. The SYN and FIN are the only controls requiring this protection, and these controls are used only at connection opening and closing. For sequence number purposes, the SYN is considered to occur before the first actual data octet of the segment in which it occurs, while the FIN is considered to occur after the last actual data octet in a segment in which it occurs. The segment length (SEG.LEN) includes both data and sequence space occupying controls. When a SYN is present then SEG.SEQ is the sequence number of the SYN.

Initial Sequence Number Selection

The protocol places no restriction on a particular connection being used over and over again. A connection is defined by a pair of sockets. New instances of a connection will be referred to as incarnations of the connection. The problem that arises from this is -- "how does the TCP identify duplicate segments from previous incarnations of the connection?" This problem becomes apparent if the connection is being opened and closed in quick succession, or if the connection breaks with loss of memory and is then reestablished.

To avoid confusion we must prevent segments from one incarnation of a connection from being used while the same sequence numbers may still be present in the network from an earlier incarnation. We want to assure this, even if a TCP crashes and loses all knowledge of the sequence numbers it has been using. When new connections are created, an initial sequence number (ISN) generator is employed which selects a new 32 bit ISN. The generator is bound to a (possibly fictitious) 32 bit clock whose low order bit is incremented roughly every 4 microseconds. Thus, the ISN cycles approximately every 4.55 hours. Since we assume that segments will stay in the network no more than the Maximum Segment Lifetime (MSL) and that the MSL is less than 4.55 hours we can reasonably assume that ISN's will be unique.

For each connection there is a send sequence number and a receive sequence number. The initial send sequence number (ISS) is chosen by the data sending TCP, and the initial receive sequence number (IRS) is learned during the connection establishing procedure.

For a connection to be established or initialized, the two TCPs must synchronize on each other's initial sequence numbers. This is done in an exchange of connection establishing segments carrying a control bit called "SYN" (for synchronize) and the initial sequence numbers. As a shorthand, segments carrying the SYN bit are also called "SYNs". Hence, the solution requires a suitable mechanism for picking an initial sequence number and a slightly involved handshake to exchange the ISN's.

The synchronization requires each side to send it's own initial sequence number and to receive a confirmation of it in acknowledgment from the other side. Each side must also receive the other side's initial sequence number and send a confirming acknowledgment.

- 1) A --> B SYN my sequence number is X
- 2) A <-- B ACK your sequence number is X
- 3) A <-- B SYN my sequence number is Y
- 4) A --> B ACK your sequence number is Y

Transmission Control Protocol Functional Specification

Because steps 2 and 3 can be combined in a single message this is called the three way (or three message) handshake.

A three way handshake is necessary because sequence numbers are not tied to a global clock in the network, and TCPs may have different mechanisms for picking the ISN's. The receiver of the first SYN has no way of knowing whether the segment was an old delayed one or not, unless it remembers the last sequence number used on the connection (which is not always possible), and so it must ask the sender to verify this SYN. The three way handshake and the advantages of a clock-driven scheme are discussed in [3].

Knowing When to Keep Quiet

To be sure that a TCP does not create a segment that carries a sequence number which may be duplicated by an old segment remaining in the network, the TCP must keep quiet for a maximum segment lifetime (MSL) before assigning any sequence numbers upon starting up or recovering from a crash in which memory of sequence numbers in use was lost. For this specification the MSL is taken to be 2 minutes. This is an engineering choice, and may be changed if experience indicates it is desirable to do so. Note that if a TCP is reinitialized in some sense, yet retains its memory of sequence numbers in use, then it need not wait at all; it must only be sure to use sequence numbers larger than those recently used.

The TCP Quiet Time Concept

This specification provides that hosts which "crash" without retaining any knowledge of the last sequence numbers transmitted on each active (i.e., not closed) connection shall delay emitting any TCP segments for at least the agreed Maximum Segment Lifetime (MSL) in the internet system of which the host is a part. In the paragraphs below, an explanation for this specification is given. TCP implementors may violate the "quiet time" restriction, but only at the risk of causing some old data to be accepted as new or new data rejected as old duplicated by some receivers in the internet system.

TCPs consume sequence number space each time a segment is formed and entered into the network output queue at a source host. The duplicate detection and sequencing algorithm in the TCP protocol relies on the unique binding of segment data to sequence space to the extent that sequence numbers will not cycle through all 2^{32} values before the segment data bound to those sequence numbers has been delivered and acknowledged by the receiver and all duplicate copies of the segments have "drained" from the internet. Without such an assumption, two distinct TCP segments could conceivably be

assigned the same or overlapping sequence numbers, causing confusion at the receiver as to which data is new and which is old. Remember that each segment is bound to as many consecutive sequence numbers as there are octets of data in the segment.

Under normal conditions, TCPs keep track of the next sequence number to emit and the oldest awaiting acknowledgment so as to avoid mistakenly using a sequence number over before its first use has been acknowledged. This alone does not guarantee that old duplicate data is drained from the net, so the sequence space has been made very large to reduce the probability that a wandering duplicate will cause trouble upon arrival. At 2 megabits/sec. it takes 4.5 hours to use up 2^{32} octets of sequence space. Since the maximum segment lifetime in the net is not likely to exceed a few tens of seconds, this is deemed ample protection for foreseeable nets, even if data rates escalate to 10's of megabits/sec. At 100 megabits/sec, the cycle time is 5.4 minutes which may be a little short, but still within reason.

The basic duplicate detection and sequencing algorithm in TCP can be defeated, however, if a source TCP does not have any memory of the sequence numbers it last used on a given connection. For example, if the TCP were to start all connections with sequence number 0, then upon crashing and restarting, a TCP might re-form an earlier connection (possibly after half-open connection resolution) and emit packets with sequence numbers identical to or overlapping with packets still in the network which were emitted on an earlier incarnation of the same connection. In the absence of knowledge about the sequence numbers used on a particular connection, the TCP specification recommends that the source delay for MSL seconds before emitting segments on the connection, to allow time for segments from the earlier connection incarnation to drain from the system.

Even hosts which can remember the time of day and used it to select initial sequence number values are not immune from this problem (i.e., even if time of day is used to select an initial sequence number for each new connection incarnation).

Suppose, for example, that a connection is opened starting with sequence number S . Suppose that this connection is not used much and that eventually the initial sequence number function ($ISN(t)$) takes on a value equal to the sequence number, say S_1 , of the last segment sent by this TCP on a particular connection. Now suppose, at this instant, the host crashes, recovers, and establishes a new incarnation of the connection. The initial sequence number chosen is $S_1 = ISN(t)$ -- last used sequence number on old incarnation of connection! If the recovery occurs quickly enough, any old

Transmission Control Protocol
Functional Specification

duplicates in the net bearing sequence numbers in the neighborhood of SI may arrive and be treated as new packets by the receiver of the new incarnation of the connection.

The problem is that the recovering host may not know for how long it crashed nor does it know whether there are still old duplicates in the system from earlier connection incarnations.

One way to deal with this problem is to deliberately delay emitting segments for one MSL after recovery from a crash- this is the "quite time" specification. Hosts which prefer to avoid waiting are willing to risk possible confusion of old and new packets at a given destination may choose not to wait for the "quite time". Implementors may provide TCP users with the ability to select on a connection by connection basis whether to wait after a crash, or may informally implement the "quite time" for all connections. Obviously, even where a user selects to "wait," this is not necessary after the host has been "up" for at least MSL seconds.

To summarize: every segment emitted occupies one or more sequence numbers in the sequence space, the numbers occupied by a segment are "busy" or "in use" until MSL seconds have passed, upon crashing a block of space-time is occupied by the octets of the last emitted segment, if a new connection is started too soon and uses any of the sequence numbers in the space-time footprint of the last segment of the previous connection incarnation, there is a potential sequence number overlap area which could cause confusion at the receiver.

3.4. Establishing a connection

The "three-way handshake" is the procedure used to establish a connection. This procedure normally is initiated by one TCP and responded to by another TCP. The procedure also works if two TCP simultaneously initiate the procedure. When simultaneous attempt occurs, each TCP receives a "SYN" segment which carries no acknowledgment after it has sent a "SYN". Of course, the arrival of an old duplicate "SYN" segment can potentially make it appear, to the recipient, that a simultaneous connection initiation is in progress. Proper use of "reset" segments can disambiguate these cases.

Several examples of connection initiation follow. Although these examples do not show connection synchronization using data-carrying segments, this is perfectly legitimate, so long as the receiving TCP doesn't deliver the data to the user until it is clear the data is valid (i.e., the data must be buffered at the receiver until the connection reaches the ESTABLISHED state). The three-way handshake reduces the possibility of false connections. It is the

implementation of a trade-off between memory and messages to provide information for this checking.

The simplest three-way handshake is shown in figure 7 below. The figures should be interpreted in the following way. Each line is numbered for reference purposes. Right arrows (-->) indicate departure of a TCP segment from TCP A to TCP B, or arrival of a segment at B from A. Left arrows (<--), indicate the reverse. Ellipsis (...) indicates a segment which is still in the network (delayed). An "XXX" indicates a segment which is lost or rejected. Comments appear in parentheses. TCP states represent the state AFTER the departure or arrival of the segment (whose contents are shown in the center of each line). Segment contents are shown in abbreviated form, with sequence number, control flags, and ACK field. Other fields such as window, addresses, lengths, and text have been left out in the interest of clarity.

TCP A		TCP B
1. CLOSED		LISTEN
2. SYN-SENT	--> <SEQ=100><CTL=SYN>	--> SYN-RECEIVED
3. ESTABLISHED	<-- <SEQ=300><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
4. ESTABLISHED	--> <SEQ=101><ACK=301><CTL=ACK>	--> ESTABLISHED
5. ESTABLISHED	--> <SEQ=101><ACK=301><CTL=ACK><DATA>	--> ESTABLISHED

Basic 3-Way Handshake for Connection Synchronization

Figure 7.

In line 2 of figure 7, TCP A begins by sending a SYN segment indicating that it will use sequence numbers starting with sequence number 100. In line 3, TCP B sends a SYN and acknowledges the SYN it received from TCP A. Note that the acknowledgment field indicates TCP B is now expecting to hear sequence 101, acknowledging the SYN which occupied sequence 100.

At line 4, TCP A responds with an empty segment containing an ACK for TCP B's SYN; and in line 5, TCP A sends some data. Note that the sequence number of the segment in line 5 is the same as in line 4 because the ACK does not occupy sequence number space (if it did, we would wind up ACKing ACK's!).

Transmission Control Protocol
Functional Specification

Simultaneous initiation is only slightly more complex, as is shown in figure 8. Each TCP cycles from CLOSED to SYN-SENT to SYN-RECEIVED to ESTABLISHED.

TCP A		TCP B
1. CLOSED		CLOSED
2. SYN-SENT	--> <SEQ=100><CTL=SYN>	...
3. SYN-RECEIVED	<-- <SEQ=300><CTL=SYN>	<-- SYN-SENT
4.	... <SEQ=100><CTL=SYN>	--> SYN-RECEIVED
5. SYN-RECEIVED	--> <SEQ=100><ACK=301><CTL=SYN,ACK>	...
6. ESTABLISHED	<-- <SEQ=300><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
7.	... <SEQ=101><ACK=301><CTL=ACK>	--> ESTABLISHED

Simultaneous Connection Synchronization

Figure 8.

The principle reason for the three-way handshake is to prevent old duplicate connection initiations from causing confusion. To deal with this, a special control message, reset, has been devised. If the receiving TCP is in a non-synchronized state (i.e., SYN-SENT, SYN-RECEIVED), it returns to LISTEN on receiving an acceptable reset. If the TCP is in one of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), it aborts the connection and informs its user. We discuss this latter case under "half-open" connections below.

TCP A		TCP B
1. CLOSED		LISTEN
2. SYN-SENT	--> <SEQ=100><CTL=SYN>	...
3. (duplicate)	... <SEQ=90><CTL=SYN>	--> SYN-RECEIVED
4. SYN-SENT	<-- <SEQ=300><ACK=91><CTL=SYN,ACK>	<-- SYN-RECEIVED
5. SYN-SENT	--> <SEQ=91><CTL=RST>	--> LISTEN
6.	... <SEQ=100><CTL=SYN>	--> SYN-RECEIVED
7. SYN-SENT	<-- <SEQ=400><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
8. ESTABLISHED	--> <SEQ=101><ACK=401><CTL=ACK>	--> ESTABLISHED

Recovery from Old Duplicate SYN

Figure 9.

As a simple example of recovery from old duplicates, consider figure 9. At line 3, an old duplicate SYN arrives at TCP B. TCP B cannot tell that this is an old duplicate, so it responds normally (line 4). TCP A detects that the ACK field is incorrect and returns a RST (reset) with its SEQ field selected to make the segment believable. TCP B, on receiving the RST, returns to the LISTEN state. When the original SYN (pun intended) finally arrives at line 6, the synchronization proceeds normally. If the SYN at line 6 had arrived before the RST, a more complex exchange might have occurred with RST's sent in both directions.

Half-Open Connections and Other Anomalies

An established connection is said to be "half-open" if one of the TCPs has closed or aborted the connection at its end without the knowledge of the other, or if the two ends of the connection have become desynchronized owing to a crash that resulted in loss of memory. Such connections will automatically become reset if an attempt is made to send data in either direction. However, half-open connections are expected to be unusual, and the recovery procedure is mildly involved.

If at site A the connection no longer exists, then an attempt by the

Transmission Control Protocol
Functional Specification

user at site B to send any data on it will result in the site B TCP receiving a reset control message. Such a message indicates to the site B TCP that something is wrong, and it is expected to abort the connection.

Assume that two user processes A and B are communicating with one another when a crash occurs causing loss of memory to A's TCP. Depending on the operating system supporting A's TCP, it is likely that some error recovery mechanism exists. When the TCP is up again, A is likely to start again from the beginning or from a recovery point. As a result, A will probably try to OPEN the connection again or try to SEND on the connection it believes open. In the latter case, it receives the error message "connection not open" from the local (A's) TCP. In an attempt to establish the connection, A's TCP will send a segment containing SYN. This scenario leads to the example shown in figure 10. After TCP A crashes, the user attempts to re-open the connection. TCP B, in the meantime, thinks the connection is open.

TCP A	TCP B
1. (CRASH)	(send 300, receive 100)
2. CLOSED	ESTABLISHED
3. SYN-SENT --> <SEQ=400><CTL=SYN>	--> (??)
4. (!!) <-- <SEQ=300><ACK=100><CTL=ACK>	<-- ESTABLISHED
5. SYN-SENT --> <SEQ=100><CTL=RST>	--> (Abort!!)
6. SYN-SENT	CLOSED
7. SYN-SENT --> <SEQ=400><CTL=SYN>	-->

Half-Open Connection Discovery

Figure 10.

When the SYN arrives at line 3, TCP B, being in a synchronized state, and the incoming segment outside the window, responds with an acknowledgment indicating what sequence it next expects to hear (ACK 100). TCP A sees that this segment does not acknowledge anything it sent and, being unsynchronized, sends a reset (RST) because it has detected a half-open connection. TCP B aborts at line 5. TCP A will

continue to try to establish the connection; the problem is now reduced to the basic 3-way handshake of figure 7.

An interesting alternative case occurs when TCP A crashes and TCP B tries to send data on what it thinks is a synchronized connection. This is illustrated in figure 11. In this case, the data arriving at TCP A from TCP B (line 2) is unacceptable because no such connection exists, so TCP A sends a RST. The RST is acceptable so TCP B processes it and aborts the connection.

TCP A	TCP B
1. (CRASH)	(send 300, receive 100)
2. (??) <-- <SEQ=300><ACK=100><DATA=10><CTL=ACK>	<-- ESTABLISHED
3. --> <SEQ=100><CTL=RST>	--> (ABORT!!)

Active Side Causes Half-Open Connection Discovery

Figure 11.

In figure 12, we find the two TCPs A and B with passive connections waiting for SYN. An old duplicate arriving at TCP B (line 2) stirs B into action. A SYN-ACK is returned (line 3) and causes TCP A to generate a RST (the ACK in line 3 is not acceptable). TCP B accepts the reset and returns to its passive LISTEN state.

TCP A	TCP B
1. LISTEN	LISTEN
2. ... <SEQ=Z><CTL=SYN>	--> SYN-RECEIVED
3. (??) <-- <SEQ=X><ACK=Z+1><CTL=SYN,ACK>	<-- SYN-RECEIVED
4. --> <SEQ=Z+1><CTL=RST>	--> (return to LISTEN!)
5. LISTEN	LISTEN

Old Duplicate SYN Initiates a Reset on two Passive Sockets

Figure 12.

A variety of other cases are possible, all of which are accounted for by the following rules for RST generation and processing.

Reset Generation

As a general rule, reset (RST) must be sent whenever a segment arrives which apparently is not intended for the current connection. A reset must not be sent if it is not clear that this is the case.

There are three groups of states:

1. If the connection does not exist (CLOSED) then a reset is sent in response to any incoming segment except another reset. In particular, SYNs addressed to a non-existent connection are rejected by this means.

If the incoming segment has an ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the CLOSED state.

2. If the connection is in any non-synchronized state (LISTEN, SYN-SENT, SYN-RECEIVED), and the incoming segment acknowledges something not yet sent (the segment carries an unacceptable ACK), or if an incoming segment has a security level or compartment which does not exactly match the level and compartment requested for the connection, a reset is sent.

If our SYN has not been acknowledged and the precedence level of the incoming segment is higher than the precedence level requested then either raise the local precedence level (if allowed by the user and the system) or send a reset; or if the precedence level of the incoming segment is lower than the precedence level requested then continue as if the precedence matched exactly (if the remote TCP cannot raise the precedence level to match ours this will be detected in the next segment it sends, and the connection will be terminated then). If our SYN has been acknowledged (perhaps in this incoming segment) the precedence level of the incoming segment must match the local precedence level exactly, if it does not a reset must be sent.

If the incoming segment has an ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the same state.

3. If the connection is in a synchronized state (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), any unacceptable segment (out of window sequence number or unacceptable acknowledgment number) must elicit only an empty acknowledgment segment containing the current send-sequence number and an acknowledgment indicating the next sequence number expected to be received, and the connection remains in the same state.

If an incoming segment has a security level, or compartment, or precedence which does not exactly match the level, and compartment, and precedence requested for the connection, a reset is sent and connection goes to the CLOSED state. The reset takes its sequence number from the ACK field of the incoming segment.

Reset Processing

In all states except SYN-SENT, all reset (RST) segments are validated by checking their SEQ-fields. A reset is valid if its sequence number is in the window. In the SYN-SENT state (a RST received in response to an initial SYN), the RST is acceptable if the ACK field acknowledges the SYN.

The receiver of a RST first validates it, then changes state. If the receiver was in the LISTEN state, it ignores it. If the receiver was in SYN-RECEIVED state and had previously been in the LISTEN state, then the receiver returns to the LISTEN state, otherwise the receiver aborts the connection and goes to the CLOSED state. If the receiver was in any other state, it aborts the connection and advises the user and goes to the CLOSED state.

3.5. Closing a Connection

CLOSE is an operation meaning "I have no more data to send." The notion of closing a full-duplex connection is subject to ambiguous interpretation, of course, since it may not be obvious how to treat the receiving side of the connection. We have chosen to treat CLOSE in a simplex fashion. The user who CLOSEs may continue to RECEIVE until he is told that the other side has CLOSED also. Thus, a program could initiate several SENDs followed by a CLOSE, and then continue to RECEIVE until signaled that a RECEIVE failed because the other side has CLOSED. We assume that the TCP will signal a user, even if no RECEIVES are outstanding, that the other side has closed, so the user can terminate his side gracefully. A TCP will reliably deliver all buffers SENT before the connection was CLOSED so a user who expects no data in return need only wait to hear the connection was CLOSED successfully to know that all his data was received at the destination TCP. Users must keep reading connections they close for sending until the TCP says no more data.

Transmission Control Protocol
Functional Specification

There are essentially three cases:

- 1) The user initiates by telling the TCP to CLOSE the connection
- 2) The remote TCP initiates by sending a FIN control signal
- 3) Both users CLOSE simultaneously

Case 1: Local user initiates the close

In this case, a FIN segment can be constructed and placed on the outgoing segment queue. No further SENDs from the user will be accepted by the TCP, and it enters the FIN-WAIT-1 state. RECEIVES are allowed in this state. All segments preceding and including FIN will be retransmitted until acknowledged. When the other TCP has both acknowledged the FIN and sent a FIN of its own, the first TCP can ACK this FIN. Note that a TCP receiving a FIN will ACK but not send its own FIN until its user has CLOSED the connection also.

Case 2: TCP receives a FIN from the network

If an unsolicited FIN arrives from the network, the receiving TCP can ACK it and tell the user that the connection is closing. The user will respond with a CLOSE, upon which the TCP can send a FIN to the other TCP after sending any remaining data. The TCP then waits until its own FIN is acknowledged whereupon it deletes the connection. If an ACK is not forthcoming, after the user timeout the connection is aborted and the user is told.

Case 3: both users close simultaneously

A simultaneous CLOSE by users at both ends of a connection causes FIN segments to be exchanged. When all segments preceding the FINs have been processed and acknowledged, each TCP can ACK the FIN it has received. Both will, upon receiving these ACKs, delete the connection.

TCP A		TCP B
1. ESTABLISHED		ESTABLISHED
2. (Close)		
FIN-WAIT-1	--> <SEQ=100><ACK=300><CTL=FIN,ACK>	--> CLOSE-WAIT
3. FIN-WAIT-2	<-- <SEQ=300><ACK=101><CTL=ACK>	<-- CLOSE-WAIT
4. TIME-WAIT	<-- <SEQ=300><ACK=101><CTL=FIN,ACK>	(Close) <-- LAST-ACK
5. TIME-WAIT	--> <SEQ=101><ACK=301><CTL=ACK>	--> CLOSED
6. (2 MSL) CLOSED		

Normal Close Sequence

Figure 13.

TCP A		TCP B
1. ESTABLISHED		ESTABLISHED
2. (Close)		(Close)
FIN-WAIT-1	--> <SEQ=100><ACK=300><CTL=FIN,ACK>	... FIN-WAIT-1
	<-- <SEQ=300><ACK=100><CTL=FIN,ACK>	<--
	... <SEQ=100><ACK=300><CTL=FIN,ACK>	-->
3. CLOSING	--> <SEQ=101><ACK=301><CTL=ACK>	... CLOSING
	<-- <SEQ=301><ACK=101><CTL=ACK>	<--
	... <SEQ=101><ACK=301><CTL=ACK>	-->
4. TIME-WAIT (2 MSL) CLOSED		TIME-WAIT (2 MSL) CLOSED

Simultaneous Close Sequence

Figure 14.

Transmission Control Protocol Functional Specification

3.6. Precedence and Security

The intent is that connection be allowed only between ports operating with exactly the same security and compartment values and at the higher of the precedence level requested by the two ports.

The precedence and security parameters used in TCP are exactly those defined in the Internet Protocol (IP) [2]. Throughout this TCP specification the term "security/compartment" is intended to indicate the security parameters used in IP including security, compartment, user group, and handling restriction.

A connection attempt with mismatched security/compartment values or a lower precedence value must be rejected by sending a reset. Rejecting a connection due to too low a precedence only occurs after an acknowledgment of the SYN has been received.

Note that TCP modules which operate only at the default value of precedence will still have to check the precedence of incoming segments and possibly raise the precedence level they use on the connection.

The security parameters may be used even in a non-secure environment (the values would indicate unclassified data), thus hosts in non-secure environments must be prepared to receive the security parameters, though they need not send them.

3.7. Data Communication

Once the connection is established data is communicated by the exchange of segments. Because segments may be lost due to errors (checksum test failure), or network congestion, TCP uses retransmission (after a timeout) to ensure delivery of every segment. Duplicate segments may arrive due to network or TCP retransmission. As discussed in the section on sequence numbers the TCP performs certain tests on the sequence and acknowledgment numbers in the segments to verify their acceptability.

The sender of data keeps track of the next sequence number to use in the variable SND.NXT. The receiver of data keeps track of the next sequence number to expect in the variable RCV.NXT. The sender of data keeps track of the oldest unacknowledged sequence number in the variable SND.UNA. If the data flow is momentarily idle and all data sent has been acknowledged then the three variables will be equal.

When the sender creates a segment and transmits it the sender advances SND.NXT. When the receiver accepts a segment it advances RCV.NXT and sends an acknowledgment. When the data sender receives an

[Page 40]

acknowledgment it advances SND.UNA. The extent to which the values of these variables differ is a measure of the delay in the communication. The amount by which the variables are advanced is the length of the data in the segment. Note that once in the ESTABLISHED state all segments must carry current acknowledgment information.

The CLOSE user call implies a push function, as does the FIN control flag in an incoming segment.

Retransmission Timeout

Because of the variability of the networks that compose an internetwork system and the wide range of uses of TCP connections the retransmission timeout must be dynamically determined. One procedure for determining a retransmission time out is given here as an illustration.

An Example Retransmission Timeout Procedure

Measure the elapsed time between sending a data octet with a particular sequence number and receiving an acknowledgment that covers that sequence number (segments sent do not have to match segments received). This measured elapsed time is the Round Trip Time (RTT). Next compute a Smoothed Round Trip Time (SRTT) as:

$$SRTT = (\text{ALPHA} * SRTT) + ((1-\text{ALPHA}) * RTT)$$

and based on this, compute the retransmission timeout (RTO) as:

$$RTO = \min[\text{UBOUND}, \max[\text{LBOUND}, (\text{BETA} * SRTT)]]$$

where UBOUND is an upper bound on the timeout (e.g., 1 minute), LBOUND is a lower bound on the timeout (e.g., 1 second), ALPHA is a smoothing factor (e.g., .8 to .9), and BETA is a delay variance factor (e.g., 1.3 to 2.0).

The Communication of Urgent Information

The objective of the TCP urgent mechanism is to allow the sending user to stimulate the receiving user to accept some urgent data and to permit the receiving TCP to indicate to the receiving user when all the currently known urgent data has been received by the user.

This mechanism permits a point in the data stream to be designated as the end of urgent information. Whenever this point is in advance of the receive sequence number (RCV.NXT) at the receiving TCP, that TCP must tell the user to go into "urgent mode"; when the receive sequence number catches up to the urgent pointer, the TCP must tell user to go

Transmission Control Protocol Functional Specification

into "normal mode". If the urgent pointer is updated while the user is in "urgent mode", the update will be invisible to the user.

The method employs a urgent field which is carried in all segments transmitted. The URG control flag indicates that the urgent field is meaningful and must be added to the segment sequence number to yield the urgent pointer. The absence of this flag indicates that there is no urgent data outstanding.

To send an urgent indication the user must also send at least one data octet. If the sending user also indicates a push, timely delivery of the urgent information to the destination process is enhanced.

Managing the Window

The window sent in each segment indicates the range of sequence numbers the sender of the window (the data receiver) is currently prepared to accept. There is an assumption that this is related to the currently available data buffer space available for this connection.

Indicating a large window encourages transmissions. If more data arrives than can be accepted, it will be discarded. This will result in excessive retransmissions, adding unnecessarily to the load on the network and the TCPs. Indicating a small window may restrict the transmission of data to the point of introducing a round trip delay between each new segment transmitted.

The mechanisms provided allow a TCP to advertise a large window and to subsequently advertise a much smaller window without having accepted that much data. This, so called "shrinking the window," is strongly discouraged. The robustness principle dictates that TCPs will not shrink the window themselves, but will be prepared for such behavior on the part of other TCPs.

The sending TCP must be prepared to accept from the user and send at least one octet of new data even if the send window is zero. The sending TCP must regularly retransmit to the receiving TCP even when the window is zero. Two minutes is recommended for the retransmission interval when the window is zero. This retransmission is essential to guarantee that when either TCP has a zero window the re-opening of the window will be reliably reported to the other.

When the receiving TCP has a zero window and a segment arrives it must still send an acknowledgment showing its next expected sequence number and current window (zero).

The sending TCP packages the data to be transmitted into segments

which fit the current window, and may repackage segments on the retransmission queue. Such repackaging is not required, but may be helpful.

In a connection with a one-way data flow, the window information will be carried in acknowledgment segments that all have the same sequence number so there will be no way to reorder them if they arrive out of order. This is not a serious problem, but it will allow the window information to be on occasion temporarily based on old reports from the data receiver. A refinement to avoid this problem is to act on the window information from segments that carry the highest acknowledgment number (that is segments with acknowledgment number equal or greater than the highest previously received).

The window management procedure has significant influence on the communication performance. The following comments are suggestions to implementers.

Window Management Suggestions

Allocating a very small window causes data to be transmitted in many small segments when better performance is achieved using fewer large segments.

One suggestion for avoiding small windows is for the receiver to defer updating a window until the additional allocation is at least X percent of the maximum allocation possible for the connection (where X might be 20 to 40).

Another suggestion is for the sender to avoid sending small segments by waiting until the window is large enough before sending data. If the the user signals a push function then the data must be sent even if it is a small segment.

Note that the acknowledgments should not be delayed or unnecessary retransmissions will result. One strategy would be to send an acknowledgment when a small segment arrives (with out updating the window information), and then to send another acknowledgment with new window information when the window is larger.

The segment sent to probe a zero window may also begin a break up of transmitted data into smaller and smaller segments. If a segment containing a single data octet sent to probe a zero window is accepted, it consumes one octet of the window now available. If the sending TCP simply sends as much as it can whenever the window is non zero, the transmitted data will be broken into alternating big and small segments. As time goes on, occasional pauses in the receiver making window allocation available will

Transmission Control Protocol Functional Specification

result in breaking the big segments into a small and not quite so big pair. And after a while the data transmission will be in mostly small segments.

The suggestion here is that the TCP implementations need to actively attempt to combine small window allocations into larger windows, since the mechanisms for managing the window tend to lead to many small windows in the simplest minded implementations.

3.8. Interfaces

There are of course two interfaces of concern: the user/TCP interface and the TCP/lower-level interface. We have a fairly elaborate model of the user/TCP interface, but the interface to the lower level protocol module is left unspecified here, since it will be specified in detail by the specification of the lower level protocol. For the case that the lower level is IP we note some of the parameter values that TCPs might use.

User/TCP Interface

The following functional description of user commands to the TCP is, at best, fictional, since every operating system will have different facilities. Consequently, we must warn readers that different TCP implementations may have different user interfaces. However, all TCPs must provide a certain minimum set of services to guarantee that all TCP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all TCP implementations.

TCP User Commands

The following sections functionally characterize a USER/TCP interface. The notation used is similar to most procedure or function calls in high level languages, but this usage is not meant to rule out trap type service calls (e.g., SVCs, UUOs, EMTs).

The user commands described below specify the basic functions the TCP must perform to support interprocess communication. Individual implementations must define their own exact format, and may provide combinations or subsets of the basic functions in single calls. In particular, some implementations may wish to automatically OPEN a connection on the first SEND or RECEIVE issued by the user for a given connection.

In providing interprocess communication facilities, the TCP must not only accept commands, but must also return information to the processes it serves. The latter consists of:

- (a) general information about a connection (e.g., interrupts, remote close, binding of unspecified foreign socket).
- (b) replies to specific user commands indicating success or various types of failure.

Open

Format: OPEN (local port, foreign socket, active/passive
[, timeout] [, precedence] [, security/compartments] [, options])
-> local connection name

We assume that the local TCP is aware of the identity of the processes it serves and will check the authority of the process to use the connection specified. Depending upon the implementation of the TCP, the local network and TCP identifiers for the source address will either be supplied by the TCP or the lower level protocol (e.g., IP). These considerations are the result of concern about security, to the extent that no TCP be able to masquerade as another one, and so on. Similarly, no process can masquerade as another without the collusion of the TCP.

If the active/passive flag is set to passive, then this is a call to LISTEN for an incoming connection. A passive open may have either a fully specified foreign socket to wait for a particular connection or an unspecified foreign socket to wait for any call. A fully specified passive call can be made active by the subsequent execution of a SEND.

A transmission control block (TCB) is created and partially filled in with data from the OPEN command parameters.

On an active OPEN command, the TCP will begin the procedure to synchronize (i.e., establish) the connection at once.

The timeout, if present, permits the caller to set up a timeout for all data submitted to TCP. If data is not successfully delivered to the destination within the timeout period, the TCP will abort the connection. The present global default is five minutes.

The TCP or some component of the operating system will verify the users authority to open a connection with the specified

Transmission Control Protocol
Functional Specification

precedence or security/compartment. The absence of precedence or security/compartment specification in the OPEN call indicates the default values must be used.

TCP will accept incoming requests as matching only if the security/compartment information is exactly the same and only if the precedence is equal to or higher than the precedence requested in the OPEN call.

The precedence for the connection is the higher of the values requested in the OPEN call and received from the incoming request, and fixed at that value for the life of the connection. Implementers may want to give the user control of this precedence negotiation. For example, the user might be allowed to specify that the precedence must be exactly matched, or that any attempt to raise the precedence be confirmed by the user.

A local connection name will be returned to the user by the TCP. The local connection name can then be used as a short hand term for the connection defined by the <local socket, foreign socket> pair.

Send

Format: SEND (local connection name, buffer address, byte count, PUSH flag, URGENT flag [,timeout])

This call causes the data contained in the indicated user buffer to be sent on the indicated connection. If the connection has not been opened, the SEND is considered an error. Some implementations may allow users to SEND first; in which case, an automatic OPEN would be done. If the calling process is not authorized to use this connection, an error is returned.

If the PUSH flag is set, the data must be transmitted promptly to the receiver, and the PUSH bit will be set in the last TCP segment created from the buffer. If the PUSH flag is not set, the data may be combined with data from subsequent SENDs for transmission efficiency.

If the URGENT flag is set, segments sent to the destination TCP will have the urgent pointer set. The receiving TCP will signal the urgent condition to the receiving process if the urgent pointer indicates that data preceding the urgent pointer has not been consumed by the receiving process. The purpose of urgent is to stimulate the receiver to process the urgent data and to indicate to the receiver when all the currently known urgent

data has been received. The number of times the sending user's TCP signals urgent will not necessarily be equal to the number of times the receiving user will be notified of the presence of urgent data.

If no foreign socket was specified in the OPEN, but the connection is established (e.g., because a LISTENing connection has become specific due to a foreign segment arriving for the local socket), then the designated buffer is sent to the implied foreign socket. Users who make use of OPEN with an unspecified foreign socket can make use of SEND without ever explicitly knowing the foreign socket address.

However, if a SEND is attempted before the foreign socket becomes specified, an error will be returned. Users can use the STATUS call to determine the status of the connection. In some implementations the TCP may notify the user when an unspecified socket is bound.

If a timeout is specified, the current user timeout for this connection is changed to the new one.

In the simplest implementation, SEND would not return control to the sending process until either the transmission was complete or the timeout had been exceeded. However, this simple method is both subject to deadlocks (for example, both sides of the connection might try to do SENDs before doing any RECEIVES) and offers poor performance, so it is not recommended. A more sophisticated implementation would return immediately to allow the process to run concurrently with network I/O, and, furthermore, to allow multiple SENDs to be in progress. Multiple SENDs are served in first come, first served order, so the TCP will queue those it cannot service immediately.

We have implicitly assumed an asynchronous user interface in which a SEND later elicits some kind of SIGNAL or pseudo-interrupt from the serving TCP. An alternative is to return a response immediately. For instance, SENDs might return immediate local acknowledgment, even if the segment sent had not been acknowledged by the distant TCP. We could optimistically assume eventual success. If we are wrong, the connection will close anyway due to the timeout. In implementations of this kind (synchronous), there will still be some asynchronous signals, but these will deal with the connection itself, and not with specific segments or buffers.

In order for the process to distinguish among error or success indications for different SENDs, it might be appropriate for the

Transmission Control Protocol
Functional Specification

buffer address to be returned along with the coded response to the SEND request. TCP-to-user signals are discussed below, indicating the information which should be returned to the calling process.

Receive

Format: RECEIVE (local connection name, buffer address, byte count) -> byte count, urgent flag, push flag

This command allocates a receiving buffer associated with the specified connection. If no OPEN precedes this command or the calling process is not authorized to use this connection, an error is returned.

In the simplest implementation, control would not return to the calling program until either the buffer was filled, or some error occurred, but this scheme is highly subject to deadlocks. A more sophisticated implementation would permit several RECEIVES to be outstanding at once. These would be filled as segments arrive. This strategy permits increased throughput at the cost of a more elaborate scheme (possibly asynchronous) to notify the calling program that a PUSH has been seen or a buffer filled.

If enough data arrive to fill the buffer before a PUSH is seen, the PUSH flag will not be set in the response to the RECEIVE. The buffer will be filled with as much data as it can hold. If a PUSH is seen before the buffer is filled the buffer will be returned partially filled and PUSH indicated.

If there is urgent data the user will have been informed as soon as it arrived via a TCP-to-user signal. The receiving user should thus be in "urgent mode". If the URGENT flag is on, additional urgent data remains. If the URGENT flag is off, this call to RECEIVE has returned all the urgent data, and the user may now leave "urgent mode". Note that data following the urgent pointer (non-urgent data) cannot be delivered to the user in the same buffer with preceeding urgent data unless the boundary is clearly marked for the user.

To distinguish among several outstanding RECEIVES and to take care of the case that a buffer is not completely filled, the return code is accompanied by both a buffer pointer and a byte count indicating the actual length of the data received.

Alternative implementations of RECEIVE might have the TCP

allocate buffer storage, or the TCP might share a ring buffer with the user.

Close

Format: CLOSE (local connection name)

This command causes the connection specified to be closed. If the connection is not open or the calling process is not authorized to use this connection, an error is returned. Closing connections is intended to be a graceful operation in the sense that outstanding SENDs will be transmitted (and retransmitted), as flow control permits, until all have been serviced. Thus, it should be acceptable to make several SEND calls, followed by a CLOSE, and expect all the data to be sent to the destination. It should also be clear that users should continue to RECEIVE on CLOSING connections, since the other side may be trying to transmit the last of its data. Thus, CLOSE means "I have no more to send" but does not mean "I will not receive any more." It may happen (if the user level protocol is not well thought out) that the closing side is unable to get rid of all its data before timing out. In this event, CLOSE turns into ABORT, and the closing TCP gives up.

The user may CLOSE the connection at any time on his own initiative, or in response to various prompts from the TCP (e.g., remote close executed, transmission timeout exceeded, destination inaccessible).

Because closing a connection requires communication with the foreign TCP, connections may remain in the closing state for a short time. Attempts to reopen the connection before the TCP replies to the CLOSE command will result in error responses.

Close also implies push function.

Status

Format: STATUS (local connection name) -> status data

This is an implementation dependent user command and could be excluded without adverse effect. Information returned would typically come from the TCB associated with the connection.

This command returns a data block containing the following information:

local socket,

Transmission Control Protocol Functional Specification

foreign socket,
local connection name,
receive window,
send window,
connection state,
number of buffers awaiting acknowledgment,
number of buffers pending receipt,
urgent state,
precedence,
security/compartments,
and transmission timeout.

Depending on the state of the connection, or on the implementation itself, some of this information may not be available or meaningful. If the calling process is not authorized to use this connection, an error is returned. This prevents unauthorized processes from gaining information about a connection.

Abort

Format: ABORT (local connection name)

This command causes all pending SENDs and RECEIVES to be aborted, the TCB to be removed, and a special RESET message to be sent to the TCP on the other side of the connection. Depending on the implementation, users may receive abort indications for each outstanding SEND or RECEIVE, or may simply receive an ABORT-acknowledgment.

TCP-to-User Messages

It is assumed that the operating system environment provides a means for the TCP to asynchronously signal the user program. When the TCP does signal a user program, certain information is passed to the user. Often in the specification the information will be an error message. In other cases there will be information relating to the completion of processing a SEND or RECEIVE or other user call.

The following information is provided:

Local Connection Name	Always
Response String	Always
Buffer Address	Send & Receive
Byte count (counts bytes received)	Receive
Push flag	Receive
Urgent flag	Receive

TCP/Lower-Level Interface

The TCP calls on a lower level protocol module to actually send and receive information over a network. One case is that of the ARPA internetwork system where the lower level module is the Internet Protocol (IP) [2].

If the lower level protocol is IP it provides arguments for a type of service and for a time to live. TCP uses the following settings for these parameters:

Type of Service = Precedence: routine, Delay: normal, Throughput: normal, Reliability: normal; or 00000000.

Time to Live = one minute, or 001111100.

Note that the assumed maximum segment lifetime is two minutes. Here we explicitly ask that a segment be destroyed if it cannot be delivered by the internet system within one minute.

If the lower level is IP (or other protocol that provides this feature) and source routing is used, the interface must allow the route information to be communicated. This is especially important so that the source and destination addresses used in the TCP checksum be the originating source and ultimate destination. It is also important to preserve the return route to answer connection requests.

Any lower level protocol will have to provide the source address, destination address, and protocol fields, and some way to determine the "TCP length", both to provide the functional equivalent service of IP and to be used in the TCP checksum.

Transmission Control Protocol Functional Specification

3.9. Event Processing

The processing depicted in this section is an example of one possible implementation. Other implementations may have slightly different processing sequences, but they should differ from those in this section only in detail, not in substance.

The activity of the TCP can be characterized as responding to events. The events that occur can be cast into three categories: user calls, arriving segments, and timeouts. This section describes the processing the TCP does in response to each of the events. In many cases the processing required depends on the state of the connection.

Events that occur:

User Calls

- OPEN
- SEND
- RECEIVE
- CLOSE
- ABORT
- STATUS

Arriving Segments

- SEGMENT ARRIVES

Timeouts

- USER TIMEOUT
- RETRANSMISSION TIMEOUT
- TIME-WAIT TIMEOUT

The model of the TCP/user interface is that user commands receive an immediate return and possibly a delayed response via an event or pseudo interrupt. In the following descriptions, the term "signal" means cause a delayed response.

Error responses are given as character strings. For example, user commands referencing connections that do not exist receive "error: connection not open".

Please note in the following that all arithmetic on sequence numbers, acknowledgment numbers, windows, et cetera, is modulo 2^{32} the size of the sequence number space. Also note that " $\leq$ " means less than or equal to (modulo 2^{32}).

[Page 52]

September 1981

Transmission Control Protocol
Functional Specification

A natural way to think about processing incoming segments is to imagine that they are first tested for proper sequence number (i.e., that their contents lie in the range of the expected "receive window" in the sequence number space) and then that they are generally queued and processed in sequence number order.

When a segment overlaps other already received segments we reconstruct the segment to contain just the new data, and adjust the header fields to be consistent.

Note that if no state change is mentioned the TCP stays in the same state.

[Page 53]

OPEN Call

CLOSED STATE (i.e., TCB does not exist)

Create a new transmission control block (TCB) to hold connection state information. Fill in local socket identifier, foreign socket, precedence, security/compartments, and user timeout information. Note that some parts of the foreign socket may be unspecified in a passive OPEN and are to be filled in by the parameters of the incoming SYN segment. Verify the security and precedence requested are allowed for this user, if not return "error: precedence not allowed" or "error: security/compartments not allowed." If passive enter the LISTEN state and return. If active and the foreign socket is unspecified, return "error: foreign socket unspecified"; if active and the foreign socket is specified, issue a SYN segment. An initial send sequence number (ISS) is selected. A SYN segment of the form <SEQ=ISS><CTL=SYN> is sent. Set SND.UNA to ISS, SND.NXT to ISS+1, enter SYN-SENT state, and return.

If the caller does not have access to the local socket specified, return "error: connection illegal for this process". If there is no room to create a new connection, return "error: insufficient resources".

LISTEN STATE

If active and the foreign socket is specified, then change the connection from passive to active, select an ISS. Send a SYN segment, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data associated with SEND may be sent with SYN segment or queued for transmission after entering ESTABLISHED state. The urgent bit if requested in the command must be sent with the data segments sent as a result of this command. If there is no room to queue the request, respond with "error: insufficient resources". If Foreign socket was not specified, then return "error: foreign socket unspecified".

September 1981

Transmission Control Protocol
Functional Specification

OPEN Call

SYN-SENT STATE
SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection already exists".

SEND Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, then return "error: connection illegal for this process".

Otherwise, return "error: connection does not exist".

LISTEN STATE

If the foreign socket is specified, then change the connection from passive to active, select an ISS. Send a SYN segment, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data associated with SEND may be sent with SYN segment or queued for transmission after entering ESTABLISHED state. The urgent bit if requested in the command must be sent with the data segments sent as a result of this command. If there is no room to queue the request, respond with "error: insufficient resources". If Foreign socket was not specified, then return "error: foreign socket unspecified".

SYN-SENT STATE

SYN-RECEIVED STATE

Queue the data for transmission after entering ESTABLISHED state. If no space to queue, respond with "error: insufficient resources".

ESTABLISHED STATE

CLOSE-WAIT STATE

Segmentize the buffer and send it with a piggybacked acknowledgment (acknowledgment value = RCV.NXT). If there is insufficient space to remember this buffer, simply return "error: insufficient resources".

If the urgent flag is set, then SND.UP <- SND.NXT-1 and set the urgent pointer in the outgoing segments.

September 1981

Transmission Control Protocol
Functional Specification

SEND Call

FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection closing" and do not service request.

RECEIVE Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, return
"error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

SYN-SENT STATE

SYN-RECEIVED STATE

Queue for processing after entering ESTABLISHED state. If there
is no room to queue this request, respond with "error:
insufficient resources".

ESTABLISHED STATE

FIN-WAIT-1 STATE

FIN-WAIT-2 STATE

If insufficient incoming segments are queued to satisfy the
request, queue the request. If there is no queue space to
remember the RECEIVE, respond with "error: insufficient
resources".

Reassemble queued incoming segments into receive buffer and return
to user. Mark "push seen" (PUSH) if this is the case.

If RCV.UP is in advance of the data currently being passed to the
user notify the user of the presence of urgent data.

When the TCP takes responsibility for delivering data to the user
that fact must be communicated to the sender via an
acknowledgment. The formation of such an acknowledgment is
described below in the discussion of processing an incoming
segment.

September 1981

Transmission Control Protocol
Functional Specification

RECEIVE Call

CLOSE-WAIT STATE

Since the remote side has already sent FIN, RECEIVES must be satisfied by text already on hand, but not yet delivered to the user. If no text is awaiting delivery, the RECEIVE will get a "error: connection closing" response. Otherwise, any remaining text can be used to satisfy the RECEIVE.

CLOSING STATE

LAST-ACK STATE

TIME-WAIT STATE

Return "error: connection closing".

CLOSE Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, return "error: connection illegal for this process".

Otherwise, return "error: connection does not exist".

LISTEN STATE

Any outstanding RECEIVES are returned with "error: closing" responses. Delete TCB, enter CLOSED state, and return.

SYN-SENT STATE

Delete the TCB and return "error: closing" responses to any queued SENDs, or RECEIVES.

SYN-RECEIVED STATE

If no SENDs have been issued and there is no pending data to send, then form a FIN segment and send it, and enter FIN-WAIT-1 state; otherwise queue for processing after entering ESTABLISHED state.

ESTABLISHED STATE

Queue this until all preceding SENDs have been segmentized, then form a FIN segment and send it. In any case, enter FIN-WAIT-1 state.

FIN-WAIT-1 STATE

FIN-WAIT-2 STATE

Strictly speaking, this is an error and should receive a "error: connection closing" response. An "ok" response would be acceptable, too, as long as a second FIN is not emitted (the first FIN may be retransmitted though).

September 1981

Transmission Control Protocol
Functional Specification

CLOSE Call

CLOSE-WAIT STATE

Queue this request until all preceding SENDs have been
segmentized; then send a FIN segment, enter CLOSING state.

CLOSING STATE

LAST-ACK STATE

TIME-WAIT STATE

Respond with "error: connection closing".

ABORT Call

CLOSED STATE (i.e., TCB does not exist)

If the user should not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

Any outstanding RECEIVES should be returned with "error: connection reset" responses. Delete TCB, enter CLOSED state, and return.

SYN-SENT STATE

All queued SENDs and RECEIVES should be given "connection reset" notification, delete the TCB, enter CLOSED state, and return.

SYN-RECEIVED STATE

ESTABLISHED STATE

FIN-WAIT-1 STATE

FIN-WAIT-2 STATE

CLOSE-WAIT STATE

Send a reset segment:

<SEQ=SEND.NXT><CTL=RST>

All queued SENDs and RECEIVES should be given "connection reset" notification; all segments queued for transmission (except for the RST formed above) or retransmission should be flushed, delete the TCB, enter CLOSED state, and return.

CLOSING STATE

LAST-ACK STATE

TIME-WAIT STATE

Respond with "ok" and delete the TCB, enter CLOSED state, and return.

STATUS Call

STATUS Call

CLOSED STATE (i.e., TCB does not exist)

If the user should not have access to such a connection, return
"error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

Return "state = LISTEN", and the TCB pointer.

SYN-SENT STATE

Return "state = SYN-SENT", and the TCB pointer.

SYN-RECEIVED STATE

Return "state = SYN-RECEIVED", and the TCB pointer.

ESTABLISHED STATE

Return "state = ESTABLISHED", and the TCB pointer.

FIN-WAIT-1 STATE

Return "state = FIN-WAIT-1", and the TCB pointer.

FIN-WAIT-2 STATE

Return "state = FIN-WAIT-2", and the TCB pointer.

CLOSE-WAIT STATE

Return "state = CLOSE-WAIT", and the TCB pointer.

CLOSING STATE

Return "state = CLOSING", and the TCB pointer.

LAST-ACK STATE

Return "state = LAST-ACK", and the TCB pointer.

TIME-WAIT STATE

Return "state = TIME-WAIT", and the TCB pointer.

SEGMENT ARRIVES

SEGMENT ARRIVES

If the state is CLOSED (i.e., TCB does not exist) then

all data in the incoming segment is discarded. An incoming segment containing a RST is discarded. An incoming segment not containing a RST causes a RST to be sent in response. The acknowledgment and sequence field values are selected to make the reset sequence acceptable to the TCP that sent the offending segment.

If the ACK bit is off, sequence number zero is used,

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If the ACK bit is on,

<SEQ=SEG.ACK><CTL=RST>

Return.

If the state is LISTEN then

first check for an RST

An incoming RST should be ignored. Return.

second check for an ACK

Any acknowledgment is bad if it arrives on a connection still in the LISTEN state. An acceptable reset segment should be formed for any arriving ACK-bearing segment. The RST should be formatted as follows:

<SEQ=SEG.ACK><CTL=RST>

Return.

third check for a SYN

If the SYN bit is set, check the security. If the security/compartment on the incoming segment does not exactly match the security/compartment in the TCB then send a reset and return.

<SEQ=SEG.ACK><CTL=RST>

SEGMENT ARRIVES

If the SEG.PRC is greater than the TCB.PRC then if allowed by the user and the system set TCB.PRC<-SEG.PRC, if not allowed send a reset and return.

<SEQ=SEG.ACK><CTL=RST>

If the SEG.PRC is less than the TCB.PRC then continue.

Set RCV.NXT to SEG.SEQ+1, IRS is set to SEG.SEQ and any other control or text should be queued for processing later. ISS should be selected and a SYN segment sent of the form:

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

SND.NXT is set to ISS+1 and SND.UNA to ISS. The connection state should be changed to SYN-RECEIVED. Note that any other incoming control or data (combined with SYN) will be processed in the SYN-RECEIVED state, but processing of SYN and ACK should not be repeated. If the listen was not fully specified (i.e., the foreign socket was not fully specified), then the unspecified fields should be filled in now.

fourth other text or control

Any other control or text-bearing segment (not containing SYN) must have an ACK and thus would be discarded by the ACK processing. An incoming RST segment could not be valid, since it could not have been sent in response to anything sent by this incarnation of the connection. So you are unlikely to get here, but if you do, drop the segment, and return.

If the state is SYN-SENT then

first check the ACK bit

If the ACK bit is set

If SEG.ACK =< ISS, or SEG.ACK > SND.NXT, send a reset (unless the RST bit is set, if so drop the segment and return)

<SEQ=SEG.ACK><CTL=RST>

and discard the segment. Return.

If SND.UNA =< SEG.ACK =< SND.NXT then the ACK is acceptable.

second check the RST bit

SEGMENT ARRIVES

If the RST bit is set

If the ACK was acceptable then signal the user "error: connection reset", drop the segment, enter CLOSED state, delete TCB, and return. Otherwise (no ACK) drop the segment and return.

third check the security and precedence

If the security/compartiment in the segment does not exactly match the security/compartiment in the TCB, send a reset

If there is an ACK

<SEQ=SEG.ACK><CTL=RST>

Otherwise

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If there is an ACK

The precedence in the segment must match the precedence in the TCB, if not, send a reset

<SEQ=SEG.ACK><CTL=RST>

If there is no ACK

If the precedence in the segment is higher than the precedence in the TCB then if allowed by the user and the system raise the precedence in the TCB to that in the segment, if not allowed to raise the prec then send a reset.

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If the precedence in the segment is lower than the precedence in the TCB continue.

If a reset was sent, discard the segment and return.

fourth check the SYN bit

This step should be reached only if the ACK is ok, or there is no ACK, and if the segment did not contain a RST.

If the SYN bit is on and the security/compartiment and precedence

SEGMENT ARRIVES

are acceptable then, RCV.NXT is set to SEG.SEQ+1, IRS is set to SEG.SEQ. SND.UNA should be advanced to equal SEG.ACK (if there is an ACK), and any segments on the retransmission queue which are thereby acknowledged should be removed.

If SND.UNA > ISS (our SYN has been ACKed), change the connection state to ESTABLISHED, form an ACK segment

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

and send it. Data or controls which were queued for transmission may be included. If there are other controls or text in the segment then continue processing at the sixth step below where the URG bit is checked, otherwise return.

Otherwise enter SYN-RECEIVED, form a SYN,ACK segment

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

and send it. If there are other controls or text in the segment, queue them for processing after the ESTABLISHED state has been reached, return.

fifth, if neither of the SYN or RST bits is set then drop the segment and return.

SEGMENT ARRIVES

Otherwise,

first check sequence number

SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Segments are processed in sequence. Initial tests on arrival are used to discard old duplicates, but further processing is done in SEG.SEQ order. If a segment's contents straddle the boundary between old and new, only the new parts should be processed.

There are four cases for the acceptability test for an incoming segment:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT ≤ SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT ≤ SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT ≤ SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

If the RCV.WND is zero, no segments will be acceptable, but special allowance should be made to accept valid ACKs, URGs and RSTs.

If an incoming segment is not acceptable, an acknowledgment should be sent in reply (unless the RST bit is set, if so drop the segment and return):

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

After sending the acknowledgment, drop the unacceptable segment and return.

SEGMENT ARRIVES

In the following it is assumed that the segment is the idealized segment that begins at RCV.NXT and does not exceed the window. One could tailor actual segments to fit this assumption by trimming off any portions that lie outside the window (including SYN and FIN), and only processing further if the segment then begins at RCV.NXT. Segments with higher beginning sequence numbers may be held for later processing.

second check the RST bit,

SYN-RECEIVED STATE

If the RST bit is set

If this connection was initiated with a passive OPEN (i.e., came from the LISTEN state), then return this connection to LISTEN state and return. The user need not be informed. If this connection was initiated with an active OPEN (i.e., came from SYN-SENT state) then the connection was refused, signal the user "connection refused". In either case, all segments on the retransmission queue should be removed. And in the active OPEN case, enter the CLOSED state and delete the TCB, and return.

ESTABLISHED

FIN-WAIT-1

FIN-WAIT-2

CLOSE-WAIT

If the RST bit is set then, any outstanding RECEIVES and SEND should receive "reset" responses. All segment queues should be flushed. Users should also receive an unsolicited general "connection reset" signal. Enter the CLOSED state, delete the TCB, and return.

CLOSING STATE

LAST-ACK STATE

TIME-WAIT

If the RST bit is set then, enter the CLOSED state, delete the TCB, and return.

SEGMENT ARRIVES

third check security and precedence

SYN-RECEIVED

If the security/compartment and precedence in the segment do not exactly match the security/compartment and precedence in the TCB then send a reset, and return.

ESTABLISHED STATE

If the security/compartment and precedence in the segment do not exactly match the security/compartment and precedence in the TCB then send a reset, any outstanding RECEIVES and SEND should receive "reset" responses. All segment queues should be flushed. Users should also receive an unsolicited general "connection reset" signal. Enter the CLOSED state, delete the TCB, and return.

Note this check is placed following the sequence check to prevent a segment from an old connection between these ports with a different security or precedence from causing an abort of the current connection.

fourth, check the SYN bit,

SYN-RECEIVED
ESTABLISHED STATE
FIN-WAIT STATE-1
FIN-WAIT STATE-2
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

If the SYN is in the window it is an error, send a reset, any outstanding RECEIVES and SEND should receive "reset" responses, all segment queues should be flushed, the user should also receive an unsolicited general "connection reset" signal, enter the CLOSED state, delete the TCB, and return.

If the SYN is not in the window this step would not be reached and an ack would have been sent in the first step (sequence number check).

fifth check the ACK field,

if the ACK bit is off drop the segment and return

if the ACK bit is on

SYN-RECEIVED STATE

If $\text{SND.UNA} \leq \text{SEG.ACK} \leq \text{SND.NXT}$ then enter ESTABLISHED state and continue processing.

If the segment acknowledgment is not acceptable, form a reset segment,

$\langle \text{SEQ} = \text{SEG.ACK} \rangle \langle \text{CTL} = \text{RST} \rangle$

and send it.

ESTABLISHED STATE

If $\text{SND.UNA} < \text{SEG.ACK} \leq \text{SND.NXT}$ then, set $\text{SND.UNA} \leftarrow \text{SEG.ACK}$. Any segments on the retransmission queue which are thereby entirely acknowledged are removed. Users should receive positive acknowledgments for buffers which have been SENT and fully acknowledged (i.e., SEND buffer should be returned with "ok" response). If the ACK is a duplicate ($\text{SEG.ACK} < \text{SND.UNA}$), it can be ignored. If the ACK acks something not yet sent ($\text{SEG.ACK} > \text{SND.NXT}$) then send an ACK, drop the segment, and return.

If $\text{SND.UNA} < \text{SEG.ACK} \leq \text{SND.NXT}$, the send window should be updated. If ($\text{SND.WL1} < \text{SEG.SEQ}$ or ($\text{SND.WL1} = \text{SEG.SEQ}$ and $\text{SND.WL2} \leq \text{SEG.ACK}$)), set $\text{SND.WND} \leftarrow \text{SEG.WND}$, set $\text{SND.WL1} \leftarrow \text{SEG.SEQ}$, and set $\text{SND.WL2} \leftarrow \text{SEG.ACK}$.

Note that SND.WND is an offset from SND.UNA , that SND.WL1 records the sequence number of the last segment used to update SND.WND , and that SND.WL2 records the acknowledgment number of the last segment used to update SND.WND . The check here prevents using old segments to update the window.

SEGMENT ARRIVES

FIN-WAIT-1 STATE

In addition to the processing for the ESTABLISHED state, if our FIN is now acknowledged then enter FIN-WAIT-2 and continue processing in that state.

FIN-WAIT-2 STATE

In addition to the processing for the ESTABLISHED state, if the retransmission queue is empty, the user's CLOSE can be acknowledged ("ok") but do not delete the TCB.

CLOSE-WAIT STATE

Do the same processing as for the ESTABLISHED state.

CLOSING STATE

In addition to the processing for the ESTABLISHED state, if the ACK acknowledges our FIN then enter the TIME-WAIT state, otherwise ignore the segment.

LAST-ACK STATE

The only thing that can arrive in this state is an acknowledgment of our FIN. If our FIN is now acknowledged, delete the TCB, enter the CLOSED state, and return.

TIME-WAIT STATE

The only thing that can arrive in this state is a retransmission of the remote FIN. Acknowledge it, and restart the 2 MSL timeout.

sixth, check the URG bit,

ESTABLISHED STATE

FIN-WAIT-1 STATE

FIN-WAIT-2 STATE

If the URG bit is set, $RCV.UP \leftarrow \max(RCV.UP, SEG.UP)$, and signal the user that the remote side has urgent data if the urgent pointer (RCV.UP) is in advance of the data consumed. If the user has already been signaled (or is still in the "urgent mode") for this continuous sequence of urgent data, do not signal the user again.

SEGMENT ARRIVES

CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT

This should not occur, since a FIN has been received from the remote side. Ignore the URG.

seventh, process the segment text,

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

Once in the ESTABLISHED state, it is possible to deliver segment text to user RECEIVE buffers. Text from segments can be moved into buffers until either the buffer is full or the segment is empty. If the segment empties and carries an PUSH flag, then the user is informed, when the buffer is returned, that a PUSH has been received.

When the TCP takes responsibility for delivering the data to the user it must also acknowledge the receipt of the data.

Once the TCP takes responsibility for the data it advances RCV.NXT over the data accepted, and adjusts RCV.WND as appropriate to the current buffer availability. The total of RCV.NXT and RCV.WND should not be reduced.

Please note the window management suggestions in section 3.7.

Send an acknowledgment of the form:

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

This acknowledgment should be piggybacked on a segment being transmitted if possible without incurring undue delay.

SEGMENT ARRIVES

CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

This should not occur, since a FIN has been received from the remote side. Ignore the segment text.

eighth, check the FIN bit,

Do not process the FIN if the state is CLOSED, LISTEN or SYN-SENT since the SEG.SEQ cannot be validated; drop the segment and return.

If the FIN bit is set, signal the user "connection closing" and return any pending RECEIVES with same message, advance RCV.NXT over the FIN, and send an acknowledgment for the FIN. Note that FIN implies PUSH for any segment text not yet delivered to the user.

SYN-RECEIVED STATE
ESTABLISHED STATE

Enter the CLOSE-WAIT state.

FIN-WAIT-1 STATE

If our FIN has been ACKed (perhaps in this segment), then enter TIME-WAIT, start the time-wait timer, turn off the other timers; otherwise enter the CLOSING state.

FIN-WAIT-2 STATE

Enter the TIME-WAIT state. Start the time-wait timer, turn off the other timers.

CLOSE-WAIT STATE

Remain in the CLOSE-WAIT state.

CLOSING STATE

Remain in the CLOSING state.

LAST-ACK STATE

Remain in the LAST-ACK state.

TIME-WAIT STATE

Remain in the TIME-WAIT state. Restart the 2 MSL time-wait
timeout.

and return.

USER TIMEOUT

USER TIMEOUT

For any state if the user timeout expires, flush all queues, signal the user "error: connection aborted due to user timeout" in general and for any outstanding calls, delete the TCB, enter the CLOSED state and return.

RETRANSMISSION TIMEOUT

For any state if the retransmission timeout expires on a segment in the retransmission queue, send the segment at the front of the retransmission queue again, reinitialize the retransmission timer, and return.

TIME-WAIT TIMEOUT

If the time-wait timeout expires on a connection delete the TCB, enter the CLOSED state and return.

GLOSSARY

1822

BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.

ACK

A control bit (acknowledge) occupying no sequence space, which indicates that the acknowledgment field of this segment specifies the next sequence number the sender of this segment is expecting to receive, hence acknowledging receipt of all previous sequence numbers.

ARPANET message

The unit of transmission between a host and an IMP in the ARPANET. The maximum size is about 1012 octets (8096 bits).

ARPANET packet

A unit of transmission used internally in the ARPANET between IMPs. The maximum size is about 126 octets (1008 bits).

connection

A logical communication path identified by a pair of sockets.

datagram

A message sent in a packet switched computer communications network.

Destination Address

The destination address, usually the network and host identifiers.

FIN

A control bit (finis) occupying one sequence number, which indicates that the sender will send no more data or control occupying sequence space.

fragment

A portion of a logical unit of data, in particular an internet fragment is a portion of an internet datagram.

FTP

A file transfer protocol.

Transmission Control Protocol
Glossary

header	Control information at the beginning of a message, segment, fragment, packet or block of data.
host	A computer. In particular a source or destination of messages from the point of view of the communication network.
Identification	An Internet Protocol field. This identifying value assigned by the sender aids in assembling the fragments of a datagram.
IMP	The Interface Message Processor, the packet switch of the ARPANET.
internet address	A source or destination address specific to the host level.
internet datagram	The unit of data exchanged between an internet module and the higher level protocol together with the internet header.
internet fragment	A portion of the data of an internet datagram with an internet header.
IP	Internet Protocol.
IRS	The Initial Receive Sequence number. The first sequence number used by the sender on a connection.
ISN	The Initial Sequence Number. The first sequence number used on a connection, (either ISS or IRS). Selected on a clock based procedure.
ISS	The Initial Send Sequence number. The first sequence number used by the sender on a connection.
leader	Control information at the beginning of a message or block of data. In particular, in the ARPANET, the control information on an ARPANET message at the host-IMP interface.

[Page 80]

left sequence

This is the next sequence number to be acknowledged by the data receiving TCP (or the lowest currently unacknowledged sequence number) and is sometimes referred to as the left edge of the send window.

local packet

The unit of transmission within a local network.

module

An implementation, usually in software, of a protocol or other procedure.

MSL

Maximum Segment Lifetime, the time a TCP segment can exist in the internetwork system. Arbitrarily defined to be 2 minutes.

octet

An eight bit byte.

Options

An Option field may contain several options, and each option may be several octets in length. The options are used primarily in testing situations; for example, to carry timestamps. Both the Internet Protocol and TCP provide for options fields.

packet

A package of data with a header which may or may not be logically complete. More often a physical packaging than a logical packaging of data.

port

The portion of a socket that specifies which logical input or output channel of a process is associated with the data.

process

A program in execution. A source or destination of data from the point of view of the TCP or other host-to-host protocol.

PUSH

A control bit occupying no sequence space, indicating that this segment contains data that must be pushed through to the receiving user.

RCV.NXT

receive next sequence number

Transmission Control Protocol
Glossary

RCV.UP	receive urgent pointer
RCV.WND	receive window
receive next sequence number	This is the next sequence number the local TCP is expecting to receive.
receive window	This represents the sequence numbers the local (receiving) TCP is willing to receive. Thus, the local TCP considers that segments overlapping the range RCV.NXT to $\text{RCV.NXT} + \text{RCV.WND} - 1$ carry acceptable data or control. Segments containing sequence numbers entirely outside of this range are considered duplicates and discarded.
RST	A control bit (reset), occupying no sequence space, indicating that the receiver should delete the connection without further interaction. The receiver can determine, based on the sequence number and acknowledgment fields of the incoming segment, whether it should honor the reset command or ignore it. In no case does receipt of a segment containing RST give rise to a RST in response.
RTP	Real Time Protocol: A host-to-host protocol for communication of time critical information.
SEG.ACK	segment acknowledgment
SEG.LEN	segment length
SEG.PRC	segment precedence value
SEG.SEQ	segment sequence
SEG.UP	segment urgent pointer field

SEG.WND

segment window field

segment

A logical unit of data, in particular a TCP segment is the unit of data transferred between a pair of TCP modules.

segment acknowledgment

The sequence number in the acknowledgment field of the arriving segment.

segment length

The amount of sequence number space occupied by a segment, including any controls which occupy sequence space.

segment sequence

The number in the sequence field of the arriving segment.

send sequence

This is the next sequence number the local (sending) TCP will use on the connection. It is initially selected from an initial sequence number curve (ISN) and is incremented for each octet of data or sequenced control transmitted.

send window

This represents the sequence numbers which the remote (receiving) TCP is willing to receive. It is the value of the window field specified in segments from the remote (data receiving) TCP. The range of new sequence numbers which may be emitted by a TCP lies between SND.NXT and $\text{SND.UNA} + \text{SND.WND} - 1$. (Retransmissions of sequence numbers between SND.UNA and SND.NXT are expected, of course.)

SND.NXT

send sequence

SND.UNA

left sequence

SND.UP

send urgent pointer

SND.WL1

segment sequence number at last window update

SND.WL2

segment acknowledgment number at last window update

Transmission Control Protocol
Glossary

SND.WND

send window

socket

An address which specifically includes a port identifier, that is, the concatenation of an Internet Address with a TCP port.

Source Address

The source address, usually the network and host identifiers.

SYN

A control bit in the incoming segment, occupying one sequence number, used at the initiation of a connection, to indicate where the sequence numbering will start.

TCB

Transmission control block, the data structure that records the state of a connection.

TCB.PRC

The precedence of the connection.

TCP

Transmission Control Protocol: A host-to-host protocol for reliable communication in internetwork environments.

TOS

Type of Service, an Internet Protocol field.

Type of Service

An Internet Protocol field which indicates the type of service for this internet fragment.

URG

A control bit (urgent), occupying no sequence space, used to indicate that the receiving user should be notified to do urgent processing as long as there is data to be consumed with sequence numbers less than the value indicated in the urgent pointer.

urgent pointer

A control field meaningful only when the URG bit is on. This field communicates the value of the urgent pointer which indicates the data octet associated with the sending user's urgent call.

REFERENCES

- [1] Cerf, V., and R. Kahn, "A Protocol for Packet Network Intercommunication", IEEE Transactions on Communications, Vol. COM-22, No. 5, pp 637-648, May 1974.
- [2] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification", RFC 791, USC/Information Sciences Institute, September 1981.
- [3] Dalal, Y. and C. Sunshine, "Connection Management in Transport Protocols", Computer Networks, Vol. 2, No. 6, pp. 454-473, December 1978.
- [4] Postel, J., "Assigned Numbers", RFC 790, USC/Information Sciences Institute, September 1981.

Network Working Group
Request for Comments: 2131
Obsoletes: 1541
Category: Standards Track

R. Droms
Bucknell University
March 1997

Dynamic Host Configuration Protocol

Status of this memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

The Dynamic Host Configuration Protocol (DHCP) provides a framework for passing configuration information to hosts on a TCPIP network. DHCP is based on the Bootstrap Protocol (BOOTP) [7], adding the capability of automatic allocation of reusable network addresses and additional configuration options [19]. DHCP captures the behavior of BOOTP relay agents [7, 21], and DHCP participants can interoperate with BOOTP participants [9].

Table of Contents

1. Introduction.	2
1.1 Changes to RFC1541.	3
1.2 Related Work.	4
1.3 Problem definition and issues	4
1.4 Requirements.	5
1.5 Terminology	6
1.6 Design goals.	6
2. Protocol Summary.	8
2.1 Configuration parameters repository	11
2.2 Dynamic allocation of network addresses	12
3. The Client-Server Protocol.	13
3.1 Client-server interaction - allocating a network address. . .	13
3.2 Client-server interaction - reusing a previously allocated network address	17
3.3 Interpretation and representation of time values.	20
3.4 Obtaining parameters with externally configured network address	20
3.5 Client parameters in DHCP	21
3.6 Use of DHCP in clients with multiple interfaces	22
3.7 When clients should use DHCP.	22
4. Specification of the DHCP client-server protocol.	22

4.1 Constructing and sending DHCP messages.	22
4.2 DHCP server administrative controls	25
4.3 DHCP server behavior.	26
4.4 DHCP client behavior.	34
5. Acknowledgments.	42
6. References	42
7. Security Considerations.	43
8. Author's Address	44
A. Host Configuration Parameters	45
List of Figures	
1. Format of a DHCP message	9
2. Format of the 'flags' field.	11
3. Timeline diagram of messages exchanged between DHCP client and servers when allocating a new network address.	15
4. Timeline diagram of messages exchanged between DHCP client and servers when reusing a previously allocated network address.	18
5. State-transition diagram for DHCP clients.	34
List of Tables	
1. Description of fields in a DHCP message.	10
2. DHCP messages.	14
3. Fields and options used by DHCP servers.	28
4. Client messages from various states.	33
5. Fields and options used by DHCP clients.	37

1. Introduction

The Dynamic Host Configuration Protocol (DHCP) provides configuration parameters to Internet hosts. DHCP consists of two components: a protocol for delivering host-specific configuration parameters from a DHCP server to a host and a mechanism for allocation of network addresses to hosts.

DHCP is built on a client-server model, where designated DHCP server hosts allocate network addresses and deliver configuration parameters to dynamically configured hosts. Throughout the remainder of this document, the term "server" refers to a host providing initialization parameters through DHCP, and the term "client" refers to a host requesting initialization parameters from a DHCP server.

A host should not act as a DHCP server unless explicitly configured to do so by a system administrator. The diversity of hardware and protocol implementations in the Internet would preclude reliable operation if random hosts were allowed to respond to DHCP requests. For example, IP requires the setting of many parameters within the protocol implementation software. Because IP can be used on many dissimilar kinds of network hardware, values for those parameters cannot be guessed or assumed to have correct defaults. Also, distributed address allocation schemes depend on a polling/defense

mechanism for discovery of addresses that are already in use. IP hosts may not always be able to defend their network addresses, so that such a distributed address allocation scheme cannot be guaranteed to avoid allocation of duplicate network addresses.

DHCP supports three mechanisms for IP address allocation. In "automatic allocation", DHCP assigns a permanent IP address to a client. In "dynamic allocation", DHCP assigns an IP address to a client for a limited period of time (or until the client explicitly relinquishes the address). In "manual allocation", a client's IP address is assigned by the network administrator, and DHCP is used simply to convey the assigned address to the client. A particular network will use one or more of these mechanisms, depending on the policies of the network administrator.

Dynamic allocation is the only one of the three mechanisms that allows automatic reuse of an address that is no longer needed by the client to which it was assigned. Thus, dynamic allocation is particularly useful for assigning an address to a client that will be connected to the network only temporarily or for sharing a limited pool of IP addresses among a group of clients that do not need permanent IP addresses. Dynamic allocation may also be a good choice for assigning an IP address to a new client being permanently connected to a network where IP addresses are sufficiently scarce that it is important to reclaim them when old clients are retired. Manual allocation allows DHCP to be used to eliminate the error-prone process of manually configuring hosts with IP addresses in environments where (for whatever reasons) it is desirable to manage IP address assignment outside of the DHCP mechanisms.

The format of DHCP messages is based on the format of BOOTP messages, to capture the BOOTP relay agent behavior described as part of the BOOTP specification [7, 21] and to allow interoperability of existing BOOTP clients with DHCP servers. Using BOOTP relay agents eliminates the necessity of having a DHCP server on each physical network segment.

1.1 Changes to RFC 1541

This document updates the DHCP protocol specification that appears in RFC1541. A new DHCP message type, DHCPINFORM, has been added; see section 3.4, 4.3 and 4.4 for details. The classing mechanism for identifying DHCP clients to DHCP servers has been extended to include "vendor" classes as defined in sections 4.2 and 4.3. The minimum lease time restriction has been removed. Finally, many editorial changes have been made to clarify the text as a result of experience gained in DHCP interoperability tests.

1.2 Related Work

There are several Internet protocols and related mechanisms that address some parts of the dynamic host configuration problem. The Reverse Address Resolution Protocol (RARP) [10] (through the extensions defined in the Dynamic RARP (DRARP) [5]) explicitly addresses the problem of network address discovery, and includes an automatic IP address assignment mechanism. The Trivial File Transfer Protocol (TFTP) [20] provides for transport of a boot image from a boot server. The Internet Control Message Protocol (ICMP) [16] provides for informing hosts of additional routers via "ICMP redirect" messages. ICMP also can provide subnet mask information through the "ICMP mask request" message and other information through the (obsolete) "ICMP information request" message. Hosts can locate routers through the ICMP router discovery mechanism [8].

BOOTP is a transport mechanism for a collection of configuration information. BOOTP is also extensible, and official extensions [17] have been defined for several configuration parameters. Morgan has proposed extensions to BOOTP for dynamic IP address assignment [15]. The Network Information Protocol (NIP), used by the Athena project at MIT, is a distributed mechanism for dynamic IP address assignment [19]. The Resource Location Protocol RLP [1] provides for location of higher level services. Sun Microsystems diskless workstations use a boot procedure that employs RARP, TFTP and an RPC mechanism called "bootparams" to deliver configuration information and operating system code to diskless hosts. (Sun Microsystems, Sun Workstation and SunOS are trademarks of Sun Microsystems, Inc.) Some Sun networks also use DRARP and an auto-installation mechanism to automate the configuration of new hosts in an existing network.

In other related work, the path minimum transmission unit (MTU) discovery algorithm can determine the MTU of an arbitrary internet path [14]. The Address Resolution Protocol (ARP) has been proposed as a transport protocol for resource location and selection [6]. Finally, the Host Requirements RFCs [3, 4] mention specific requirements for host reconfiguration and suggest a scenario for initial configuration of diskless hosts.

1.3 Problem definition and issues

DHCP is designed to supply DHCP clients with the configuration parameters defined in the Host Requirements RFCs. After obtaining parameters via DHCP, a DHCP client should be able to exchange packets with any other host in the Internet. The TCP/IP stack parameters supplied by DHCP are listed in Appendix A.

Not all of these parameters are required for a newly initialized client. A client and server may negotiate for the transmission of only those parameters required by the client or specific to a particular subnet.

DHCP allows but does not require the configuration of client parameters not directly related to the IP protocol. DHCP also does not address registration of newly configured clients with the Domain Name System (DNS) [12, 13].

DHCP is not intended for use in configuring routers.

1.4 Requirements

Throughout this document, the words that are used to define the significance of particular requirements are capitalized. These words are:

- o "MUST"

This word or the adjective "REQUIRED" means that the item is an absolute requirement of this specification.

- o "MUST NOT"

This phrase means that the item is an absolute prohibition of this specification.

- o "SHOULD"

This word or the adjective "RECOMMENDED" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.

- o "SHOULD NOT"

This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

- o "MAY"

This word or the adjective "OPTIONAL" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

1.5 Terminology

This document uses the following terms:

- o "DHCP client"

A DHCP client is an Internet host using DHCP to obtain configuration parameters such as a network address.

- o "DHCP server"

A DHCP server is an Internet host that returns configuration parameters to DHCP clients.

- o "BOOTP relay agent"

A BOOTP relay agent or relay agent is an Internet host or router that passes DHCP messages between DHCP clients and DHCP servers. DHCP is designed to use the same relay agent behavior as specified in the BOOTP protocol specification.

- o "binding"

A binding is a collection of configuration parameters, including at least an IP address, associated with or "bound to" a DHCP client. Bindings are managed by DHCP servers.

1.6 Design goals

The following list gives general design goals for DHCP.

- o DHCP should be a mechanism rather than a policy. DHCP must allow local system administrators control over configuration parameters where desired; e.g., local system administrators should be able to enforce local policies concerning allocation and access to local resources where desired.

- o Clients should require no manual configuration. Each client should be able to discover appropriate local configuration parameters without user intervention and incorporate those parameters into its own configuration.
- o Networks should require no manual configuration for individual clients. Under normal circumstances, the network manager should not have to enter any per-client configuration parameters.
- o DHCP should not require a server on each subnet. To allow for scale and economy, DHCP must work across routers or through the intervention of BOOTP relay agents.
- o A DHCP client must be prepared to receive multiple responses to a request for configuration parameters. Some installations may include multiple, overlapping DHCP servers to enhance reliability and increase performance.
- o DHCP must coexist with statically configured, non-participating hosts and with existing network protocol implementations.
- o DHCP must interoperate with the BOOTP relay agent behavior as described by RFC 951 and by RFC 1542 [21].
- o DHCP must provide service to existing BOOTP clients.

The following list gives design goals specific to the transmission of the network layer parameters. DHCP must:

- o Guarantee that any specific network address will not be in use by more than one DHCP client at a time,
- o Retain DHCP client configuration across DHCP client reboot. A DHCP client should, whenever possible, be assigned the same configuration parameters (e.g., network address) in response to each request,
- o Retain DHCP client configuration across server reboots, and, whenever possible, a DHCP client should be assigned the same configuration parameters despite restarts of the DHCP mechanism,
- o Allow automated assignment of configuration parameters to new clients to avoid hand configuration for new clients,
- o Support fixed or permanent allocation of configuration parameters to specific clients.

2. Protocol Summary

From the client's point of view, DHCP is an extension of the BOOTP mechanism. This behavior allows existing BOOTP clients to interoperate with DHCP servers without requiring any change to the clients' initialization software. RFC 1542 [2] details the interactions between BOOTP and DHCP clients and servers [9]. There are some new, optional transactions that optimize the interaction between DHCP clients and servers that are described in sections 3 and 4.

Figure 1 gives the format of a DHCP message and table 1 describes each of the fields in the DHCP message. The numbers in parentheses indicate the size of each field in octets. The names for the fields given in the figure will be used throughout this document to refer to the fields in DHCP messages.

There are two primary differences between DHCP and BOOTP. First, DHCP defines mechanisms through which clients can be assigned a network address for a finite lease, allowing for serial reassignment of network addresses to different clients. Second, DHCP provides the mechanism for a client to acquire all of the IP configuration parameters that it needs in order to operate.

DHCP introduces a small change in terminology intended to clarify the meaning of one of the fields. What was the "vendor extensions" field in BOOTP has been re-named the "options" field in DHCP. Similarly, the tagged data items that were used inside the BOOTP "vendor extensions" field, which were formerly referred to as "vendor extensions," are now termed simply "options."

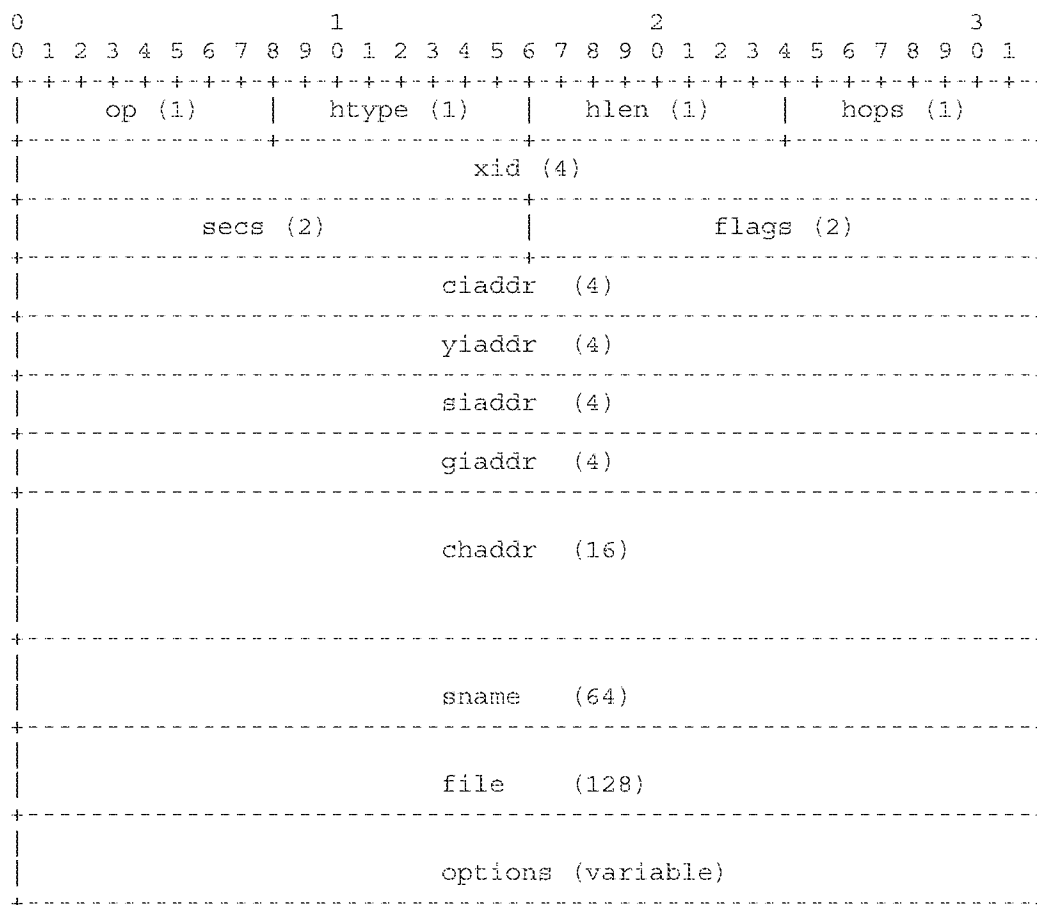


Figure 1: Format of a DHCP message

DHCP defines a new 'client identifier' option that is used to pass an explicit client identifier to a DHCP server. This change eliminates the overloading of the 'chaddr' field in BOOTP messages, where 'chaddr' is used both as a hardware address for transmission of BOOTP reply messages and as a client identifier. The 'client identifier' is an opaque key, not to be interpreted by the server; for example, the 'client identifier' may contain a hardware address, identical to the contents of the 'chaddr' field, or it may contain another type of identifier, such as a DNS name. The 'client identifier' chosen by a DHCP client MUST be unique to that client within the subnet to which the client is attached. If the client uses a 'client identifier' in one message, it MUST use that same identifier in all subsequent messages, to ensure that all servers correctly identify the client.

DHCP clarifies the interpretation of the 'siaddr' field as the address of the server to use in the next step of the client's bootstrap process. A DHCP server may return its own address in the 'siaddr' field, if the server is prepared to supply the next bootstrap service (e.g., delivery of an operating system executable image). A DHCP server always returns its own address in the 'server identifier' option.

FIELD	OCTETS	DESCRIPTION
-----	-----	-----
op	1	Message op code / message type. 1 = BOOTREQUEST, 2 = BOOTREPLY
htype	1	Hardware address type, see ARP section in "Assigned Numbers" RFC; e.g., '1' = 10mb ethernet.
hlen	1	Hardware address length (e.g. '6' for 10mb ethernet).
hops	1	Client sets to zero, optionally used by relay agents when booting via a relay agent.
xid	4	Transaction ID, a random number chosen by the client, used by the client and server to associate messages and responses between a client and a server.
secs	2	Filled in by client, seconds elapsed since client began address acquisition or renewal process.
flags	2	Flags (see figure 2).
ciaddr	4	Client IP address; only filled in if client is in BOUND, RENEW or REBINDING state and can respond to ARP requests.
yiaddr	4	'your' (client) IP address.
siaddr	4	IP address of next server to use in bootstrap; returned in DHCP OFFER, DHCPACK by server.
giaddr	4	Relay agent IP address, used in booting via a relay agent.
chaddr	16	Client hardware address.
sname	64	Optional server host name, null terminated string.
file	128	Boot file name, null terminated string; "generic" name or null in DHCPDISCOVER, fully qualified directory-path name in DHCPOFFER.
options	var	Optional parameters field. See the options documents for a list of defined options.

Table 1: Description of fields in a DHCP message

The 'options' field is now variable length. A DHCP client must be prepared to receive DHCP messages with an 'options' field of at least length 312 octets. This requirement implies that a DHCP client must be prepared to receive a message of up to 576 octets, the minimum IP

datagram size an IP host must be prepared to accept [3]. DHCP clients may negotiate the use of larger DHCP messages through the 'maximum DHCP message size' option. The options field may be further extended into the 'file' and 'sname' fields.

In the case of a client using DHCP for initial configuration (before the client's TCP/IP software has been completely configured), DHCP requires creative use of the client's TCP/IP software and liberal interpretation of RFC 1122. The TCP/IP software SHOULD accept and forward to the IP layer any IP packets delivered to the client's hardware address before the IP address is configured; DHCP servers and BOOTP relay agents may not be able to deliver DHCP messages to clients that cannot accept hardware unicast datagrams before the TCP/IP software is configured.

To work around some clients that cannot accept IP unicast datagrams before the TCP/IP software is configured as discussed in the previous paragraph, DHCP uses the 'flags' field [21]. The leftmost bit is defined as the BROADCAST (B) flag. The semantics of this flag are discussed in section 4.1 of this document. The remaining bits of the flags field are reserved for future use. They MUST be set to zero by clients and ignored by servers and relay agents. Figure 2 gives the format of the 'flags' field.

```

                                1 1 1 1 1 1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+++++
|B|                MBZ                |
+++++
```

B: BROADCAST flag

MBZ: MUST BE ZERO (reserved for future use)

Figure 2: Format of the 'flags' field

2.1 Configuration parameters repository

The first service provided by DHCP is to provide persistent storage of network parameters for network clients. The model of DHCP persistent storage is that the DHCP service stores a key-value entry for each client, where the key is some unique identifier (for example, an IP subnet number and a unique identifier within the subnet) and the value contains the configuration parameters for the client.

For example, the key might be the pair (IP-subnet-number, hardware-address) (note that the "hardware-address" should be typed by the

type of hardware to accommodate possible duplication of hardware addresses resulting from bit-ordering problems in a mixed-media, bridged network) allowing for serial or concurrent reuse of a hardware address on different subnets, and for hardware addresses that may not be globally unique. Alternately, the key might be the pair (IP-subnet-number, hostname), allowing the server to assign parameters intelligently to a DHCP client that has been moved to a different subnet or has changed hardware addresses (perhaps because the network interface failed and was replaced). The protocol defines that the key will be (IP-subnet-number, hardware-address) unless the client explicitly supplies an identifier using the 'client identifier' option. A client can query the DHCP service to retrieve its configuration parameters. The client interface to the configuration parameters repository consists of protocol messages to request configuration parameters and responses from the server carrying the configuration parameters.

2.2 Dynamic allocation of network addresses

The second service provided by DHCP is the allocation of temporary or permanent network (IP) addresses to clients. The basic mechanism for the dynamic allocation of network addresses is simple: a client requests the use of an address for some period of time. The allocation mechanism (the collection of DHCP servers) guarantees not to reallocate that address within the requested time and attempts to return the same network address each time the client requests an address. In this document, the period over which a network address is allocated to a client is referred to as a "lease" [11]. The client may extend its lease with subsequent requests. The client may issue a message to release the address back to the server when the client no longer needs the address. The client may ask for a permanent assignment by asking for an infinite lease. Even when assigning "permanent" addresses, a server may choose to give out lengthy but non-infinite leases to allow detection of the fact that the client has been retired.

In some environments it will be necessary to reassign network addresses due to exhaustion of available addresses. In such environments, the allocation mechanism will reuse addresses whose lease has expired. The server should use whatever information is available in the configuration information repository to choose an address to reuse. For example, the server may choose the least recently assigned address. As a consistency check, the allocating server SHOULD probe the reused address before allocating the address, e.g., with an ICMP echo request, and the client SHOULD probe the newly received address, e.g., with ARP.

3. The Client-Server Protocol

DHCP uses the BOOTP message format defined in RFC 951 and given in table 1 and figure 1. The 'op' field of each DHCP message sent from a client to a server contains BOOTREQUEST. BOOTREPLY is used in the 'op' field of each DHCP message sent from a server to a client.

The first four octets of the 'options' field of the DHCP message contain the (decimal) values 99, 130, 83 and 99, respectively (this is the same magic cookie as is defined in RFC 1497 [17]). The remainder of the 'options' field consists of a list of tagged parameters that are called "options". All of the "vendor extensions" listed in RFC 1497 are also DHCP options. RFC 1533 gives the complete set of options defined for use with DHCP.

Several options have been defined so far. One particular option - the "DHCP message type" option - must be included in every DHCP message. This option defines the "type" of the DHCP message. Additional options may be allowed, required, or not allowed, depending on the DHCP message type.

Throughout this document, DHCP messages that include a 'DHCP message type' option will be referred to by the type of the message; e.g., a DHCP message with 'DHCP message type' option type 1 will be referred to as a "DHCPDISCOVER" message.

3.1 Client-server interaction - allocating a network address

The following summary of the protocol exchanges between clients and servers refers to the DHCP messages described in table 2. The timeline diagram in figure 3 shows the timing relationships in a typical client-server interaction. If the client already knows its address, some steps may be omitted; this abbreviated interaction is described in section 3.2.

1. The client broadcasts a DHCPDISCOVER message on its local physical subnet. The DHCPDISCOVER message MAY include options that suggest values for the network address and lease duration. BOOTP relay agents may pass the message on to DHCP servers not on the same physical subnet.
2. Each server may respond with a DHCPOFFER message that includes an available network address in the 'yiaddr' field (and other configuration parameters in DHCP options). Servers need not reserve the offered network address, although the protocol will work more efficiently if the server avoids allocating the offered network address to another client. When allocating a new address, servers SHOULD check that the offered network address is not

already in use; e.g., the server may probe the offered address with an ICMP Echo Request. Servers SHOULD be implemented so that network administrators MAY choose to disable probes of newly allocated addresses. The server transmits the DHCP OFFER message to the client, using the BOOTP relay agent if necessary.

Message	Use
-----	---
DHCPDISCOVER	- Client broadcast to locate available servers.
DHCPOFFER	- Server to client in response to DHCPDISCOVER with offer of configuration parameters.
DHCPREQUEST	- Client message to servers either (a) requesting offered parameters from one server and implicitly declining offers from all others, (b) confirming correctness of previously allocated address after, e.g., system reboot, or (c) extending the lease on a particular network address.
DHCPACK	- Server to client with configuration parameters, including committed network address.
DHCPNAK	- Server to client indicating client's notion of network address is incorrect (e.g., client has moved to new subnet) or client's lease as expired
DHCPDECLINE	- Client to server indicating network address is already in use.
DHCPRELEASE	- Client to server relinquishing network address and cancelling remaining lease.
DHCPINFORM	- Client to server, asking only for local configuration parameters; client already has externally configured network address.

Table 2: DHCP messages

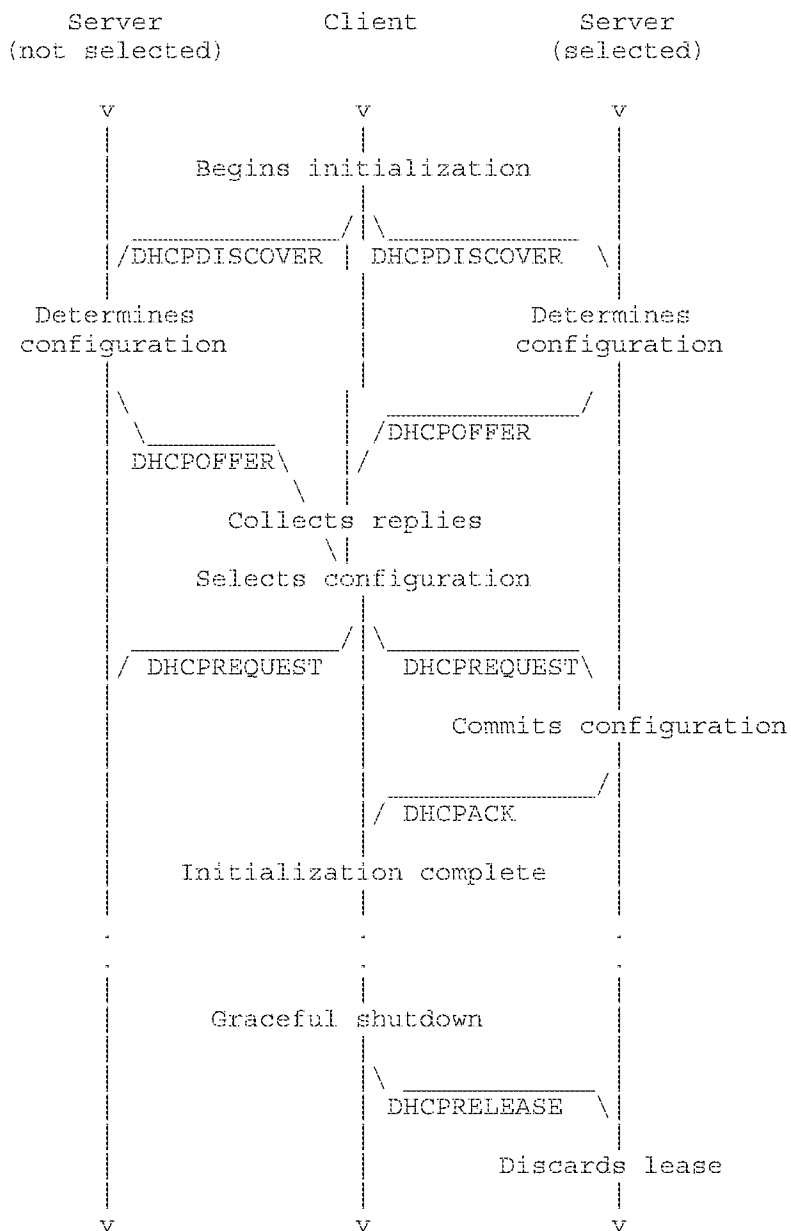


Figure 3: Timeline diagram of messages exchanged between DHCP client and servers when allocating a new network address

3. The client receives one or more DHCPOFFER messages from one or more servers. The client may choose to wait for multiple responses. The client chooses one server from which to request configuration parameters, based on the configuration parameters offered in the DHCPOFFER messages. The client broadcasts a DHCPREQUEST message that MUST include the 'server identifier' option to indicate which server it has selected, and that MAY include other options specifying desired configuration values. The 'requested IP address' option MUST be set to the value of 'yiaddr' in the DHCPOFFER message from the server. This DHCPREQUEST message is broadcast and relayed through DHCP/BOOTP relay agents. To help ensure that any BOOTP relay agents forward the DHCPREQUEST message to the same set of DHCP servers that received the original DHCPDISCOVER message, the DHCPREQUEST message MUST use the same value in the DHCP message header's 'secs' field and be sent to the same IP broadcast address as the original DHCPDISCOVER message. The client times out and retransmits the DHCPDISCOVER message if the client receives no DHCPOFFER messages.
4. The servers receive the DHCPREQUEST broadcast from the client. Those servers not selected by the DHCPREQUEST message use the message as notification that the client has declined that server's offer. The server selected in the DHCPREQUEST message commits the binding for the client to persistent storage and responds with a DHCPACK message containing the configuration parameters for the requesting client. The combination of 'client identifier' or 'chaddr' and assigned network address constitute a unique identifier for the client's lease and are used by both the client and server to identify a lease referred to in any DHCP messages. Any configuration parameters in the DHCPACK message SHOULD NOT conflict with those in the earlier DHCPOFFER message to which the client is responding. The server SHOULD NOT check the offered network address at this point. The 'yiaddr' field in the DHCPACK messages is filled in with the selected network address.

If the selected server is unable to satisfy the DHCPREQUEST message (e.g., the requested network address has been allocated), the server SHOULD respond with a DHCPNAK message.

A server MAY choose to mark addresses offered to clients in DHCPOFFER messages as unavailable. The server SHOULD mark an address offered to a client in a DHCPOFFER message as available if the server receives no DHCPREQUEST message from that client.

5. The client receives the DHCPACK message with configuration parameters. The client SHOULD perform a final check on the parameters (e.g., ARP for allocated network address), and notes the duration of the lease specified in the DHCPACK message. At this

point, the client is configured. If the client detects that the address is already in use (e.g., through the use of ARP), the client MUST send a DHCPDECLINE message to the server and restarts the configuration process. The client SHOULD wait a minimum of ten seconds before restarting the configuration process to avoid excessive network traffic in case of looping.

If the client receives a DHCPNAK message, the client restarts the configuration process.

The client times out and retransmits the DHCPREQUEST message if the client receives neither a DHCPACK or a DHCPNAK message. The client retransmits the DHCPREQUEST according to the retransmission algorithm in section 4.1. The client should choose to retransmit the DHCPREQUEST enough times to give adequate probability of contacting the server without causing the client (and the user of that client) to wait overly long before giving up; e.g., a client retransmitting as described in section 4.1 might retransmit the DHCPREQUEST message four times, for a total delay of 60 seconds, before restarting the initialization procedure. If the client receives neither a DHCPACK or a DHCPNAK message after employing the retransmission algorithm, the client reverts to INIT state and restarts the initialization process. The client SHOULD notify the user that the initialization process has failed and is restarting.

6. The client may choose to relinquish its lease on a network address by sending a DHCPRELEASE message to the server. The client identifies the lease to be released with its 'client identifier', or 'chaddr' and network address in the DHCPRELEASE message. If the client used a 'client identifier' when it obtained the lease, it MUST use the same 'client identifier' in the DHCPRELEASE message.

3.2 Client-server interaction - reusing a previously allocated network address

If a client remembers and wishes to reuse a previously allocated network address, a client may choose to omit some of the steps described in the previous section. The timeline diagram in figure 4 shows the timing relationships in a typical client-server interaction for a client reusing a previously allocated network address.

1. The client broadcasts a DHCPREQUEST message on its local subnet. The message includes the client's network address in the 'requested IP address' option. As the client has not received its network address, it MUST NOT fill in the 'ciaddr' field. BOOTP relay agents pass the message on to DHCP servers not on the same subnet. If the client used a 'client identifier' to obtain its address, the client MUST use the same 'client identifier' in the DHCPREQUEST message.
2. Servers with knowledge of the client's configuration parameters respond with a DHCPACK message to the client. Servers SHOULD NOT check that the client's network address is already in use; the client may respond to ICMP Echo Request messages at this point.

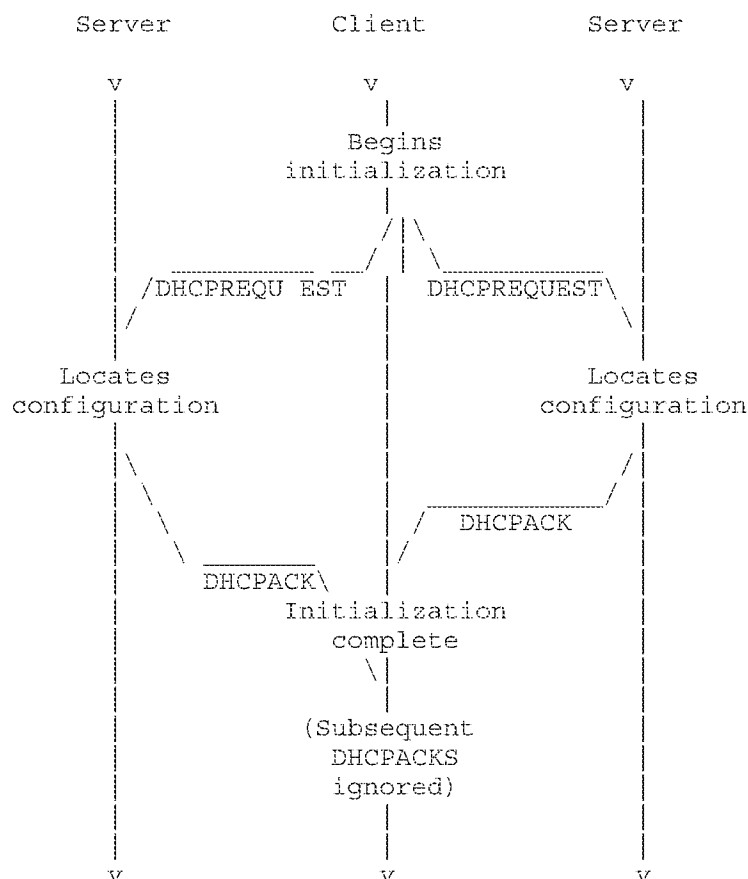


Figure 4: Timeline diagram of messages exchanged between DHCP client and servers when reusing a previously allocated network address

If the client's request is invalid (e.g., the client has moved to a new subnet), servers SHOULD respond with a DHCPNAK message to the client. Servers SHOULD NOT respond if their information is not guaranteed to be accurate. For example, a server that identifies a request for an expired binding that is owned by another server SHOULD NOT respond with a DHCPNAK unless the servers are using an explicit mechanism to maintain coherency among the servers.

If 'giaddr' is 0x0 in the DHCPREQUEST message, the client is on the same subnet as the server. The server MUST broadcast the DHCPNAK message to the 0xffffffff broadcast address because the client may not have a correct network address or subnet mask, and the client may not be answering ARP requests. Otherwise, the server MUST send the DHCPNAK message to the IP address of the BOOTP relay agent, as recorded in 'giaddr'. The relay agent will, in turn, forward the message directly to the client's hardware address, so that the DHCPNAK can be delivered even if the client has moved to a new network.

3. The client receives the DHCPACK message with configuration parameters. The client performs a final check on the parameters (as in section 3.1), and notes the duration of the lease specified in the DHCPACK message. The specific lease is implicitly identified by the 'client identifier' or 'chaddr' and the network address. At this point, the client is configured.

If the client detects that the IP address in the DHCPACK message is already in use, the client MUST send a DHCPDECLINE message to the server and restarts the configuration process by requesting a new network address. This action corresponds to the client moving to the INIT state in the DHCP state diagram, which is described in section 4.4.

If the client receives a DHCPNAK message, it cannot reuse its remembered network address. It must instead request a new address by restarting the configuration process, this time using the (non-abbreviated) procedure described in section 3.1. This action also corresponds to the client moving to the INIT state in the DHCP state diagram.

The client times out and retransmits the DHCPREQUEST message if the client receives neither a DHCPACK nor a DHCPNAK message. The client retransmits the DHCPREQUEST according to the retransmission algorithm in section 4.1. The client should choose to retransmit the DHCPREQUEST enough times to give adequate probability of contacting the server without causing the client (and the user of that client) to wait overly long before giving up; e.g., a client retransmitting as described in section 4.1 might retransmit the

DHCPREQUEST message four times, for a total delay of 60 seconds, before restarting the initialization procedure. If the client receives neither a DHCPACK or a DHCPNAK message after employing the retransmission algorithm, the client MAY choose to use the previously allocated network address and configuration parameters for the remainder of the unexpired lease. This corresponds to moving to BOUND state in the client state transition diagram shown in figure 5.

4. The client may choose to relinquish its lease on a network address by sending a DHCPRELEASE message to the server. The client identifies the lease to be released with its 'client identifier', or 'chaddr' and network address in the DHCPRELEASE message.

Note that in this case, where the client retains its network address locally, the client will not normally relinquish its lease during a graceful shutdown. Only in the case where the client explicitly needs to relinquish its lease, e.g., the client is about to be moved to a different subnet, will the client send a DHCPRELEASE message.

3.3 Interpretation and representation of time values

A client acquires a lease for a network address for a fixed period of time (which may be infinite). Throughout the protocol, times are to be represented in units of seconds. The time value of 0xffffffff is reserved to represent "infinity".

As clients and servers may not have synchronized clocks, times are represented in DHCP messages as relative times, to be interpreted with respect to the client's local clock. Representing relative times in units of seconds in an unsigned 32 bit word gives a range of relative times from 0 to approximately 100 years, which is sufficient for the relative times to be measured using DHCP.

The algorithm for lease duration interpretation given in the previous paragraph assumes that client and server clocks are stable relative to each other. If there is drift between the two clocks, the server may consider the lease expired before the client does. To compensate, the server may return a shorter lease duration to the client than the server commits to its local database of client information.

3.4 Obtaining parameters with externally configured network address

If a client has obtained a network address through some other means (e.g., manual configuration), it may use a DHCPINFORM request message

to obtain other local configuration parameters. Servers receiving a DHCPINFORM message construct a DHCPACK message with any local configuration parameters appropriate for the client without: allocating a new address, checking for an existing binding, filling in 'yiaddr' or including lease time parameters. The servers SHOULD unicast the DHCPACK reply to the address given in the 'ciaddr' field of the DHCPINFORM message.

The server SHOULD check the network address in a DHCPINFORM message for consistency, but MUST NOT check for an existing lease. The server forms a DHCPACK message containing the configuration parameters for the requesting client and sends the DHCPACK message directly to the client.

3.5 Client parameters in DHCP

Not all clients require initialization of all parameters listed in Appendix A. Two techniques are used to reduce the number of parameters transmitted from the server to the client. First, most of the parameters have defaults defined in the Host Requirements RFCs; if the client receives no parameters from the server that override the defaults, a client uses those default values. Second, in its initial DHCPDISCOVER or DHCPREQUEST message, a client may provide the server with a list of specific parameters the client is interested in. If the client includes a list of parameters in a DHCPDISCOVER message, it MUST include that list in any subsequent DHCPREQUEST messages.

The client SHOULD include the 'maximum DHCP message size' option to let the server know how large the server may make its DHCP messages. The parameters returned to a client may still exceed the space allocated to options in a DHCP message. In this case, two additional options flags (which must appear in the 'options' field of the message) indicate that the 'file' and 'sname' fields are to be used for options.

The client can inform the server which configuration parameters the client is interested in by including the 'parameter request list' option. The data portion of this option explicitly lists the options requested by tag number.

In addition, the client may suggest values for the network address and lease time in the DHCPDISCOVER message. The client may include the 'requested IP address' option to suggest that a particular IP address be assigned, and may include the 'IP address lease time' option to suggest the lease time it would like. Other options representing "hints" at configuration parameters are allowed in a DHCPDISCOVER or DHCPREQUEST message. However, additional options may

be ignored by servers, and multiple servers may, therefore, not return identical values for some options. The 'requested IP address' option is to be filled in only in a DHCPREQUEST message when the client is verifying network parameters obtained previously. The client fills in the 'ciaddr' field only when correctly configured with an IP address in BOUND, RENEWING or REBINDING state.

If a server receives a DHCPREQUEST message with an invalid 'requested IP address', the server SHOULD respond to the client with a DHCPNAK message and may choose to report the problem to the system administrator. The server may include an error message in the 'message' option.

3.6 Use of DHCP in clients with multiple interfaces

A client with multiple network interfaces must use DHCP through each interface independently to obtain configuration information parameters for those separate interfaces.

3.7 When clients should use DHCP

A client SHOULD use DHCP to reacquire or verify its IP address and network parameters whenever the local network parameters may have changed; e.g., at system boot time or after a disconnection from the local network, as the local network configuration may change without the client's or user's knowledge.

If a client has knowledge of a previous network address and is unable to contact a local DHCP server, the client may continue to use the previous network address until the lease for that address expires. If the lease expires before the client can contact a DHCP server, the client must immediately discontinue use of the previous network address and may inform local users of the problem.

4. Specification of the DHCP client-server protocol

In this section, we assume that a DHCP server has a block of network addresses from which it can satisfy requests for new addresses. Each server also maintains a database of allocated addresses and leases in local permanent storage.

4.1 Constructing and sending DHCP messages

DHCP clients and servers both construct DHCP messages by filling in fields in the fixed format section of the message and appending tagged data items in the variable length option area. The options area includes first a four-octet 'magic cookie' (which was described in section 3), followed by the options. The last option must always

be the 'end' option.

DHCP uses UDP as its transport protocol. DHCP messages from a client to a server are sent to the 'DHCP server' port (67), and DHCP messages from a server to a client are sent to the 'DHCP client' port (68). A server with multiple network address (e.g., a multi-homed host) MAY use any of its network addresses in outgoing DHCP messages.

The 'server identifier' field is used both to identify a DHCP server in a DHCP message and as a destination address from clients to servers. A server with multiple network addresses MUST be prepared to accept any of its network addresses as identifying that server in a DHCP message. To accommodate potentially incomplete network connectivity, a server MUST choose an address as a 'server identifier' that, to the best of the server's knowledge, is reachable from the client. For example, if the DHCP server and the DHCP client are connected to the same subnet (i.e., the 'giaddr' field in the message from the client is zero), the server SHOULD select the IP address the server is using for communication on that subnet as the 'server identifier'. If the server is using multiple IP addresses on that subnet, any such address may be used. If the server has received a message through a DHCP relay agent, the server SHOULD choose an address from the interface on which the message was received as the 'server identifier' (unless the server has other, better information on which to make its choice). DHCP clients MUST use the IP address provided in the 'server identifier' option for any unicast requests to the DHCP server.

DHCP messages broadcast by a client prior to that client obtaining its IP address must have the source address field in the IP header set to 0.

If the 'giaddr' field in a DHCP message from a client is non-zero, the server sends any return messages to the 'DHCP server' port on the BOOTP relay agent whose address appears in 'giaddr'. If the 'giaddr' field is zero and the 'ciaddr' field is nonzero, then the server unicasts DHCP OFFER and DHCP ACK messages to the address in 'ciaddr'. If 'giaddr' is zero and 'ciaddr' is zero, and the broadcast bit is set, then the server broadcasts DHCP OFFER and DHCP ACK messages to 0xffffffff. If the broadcast bit is not set and 'giaddr' is zero and 'ciaddr' is zero, then the server unicasts DHCP OFFER and DHCP ACK messages to the client's hardware address and 'yiaddr' address. In all cases, when 'giaddr' is zero, the server broadcasts any DHCP NAK messages to 0xffffffff.

If the options in a DHCP message extend into the 'sname' and 'file' fields, the 'option overload' option MUST appear in the 'options' field, with value 1, 2 or 3, as specified in RFC 1533. If the

'option overload' option is present in the 'options' field, the options in the 'options' field MUST be terminated by an 'end' option, and MAY contain one or more 'pad' options to fill the options field. The options in the 'sname' and 'file' fields (if in use as indicated by the 'options overload' option) MUST begin with the first octet of the field, MUST be terminated by an 'end' option, and MUST be followed by 'pad' options to fill the remainder of the field. Any individual option in the 'options', 'sname' and 'file' fields MUST be entirely contained in that field. The options in the 'options' field MUST be interpreted first, so that any 'option overload' options may be interpreted. The 'file' field MUST be interpreted next (if the 'option overload' option indicates that the 'file' field contains DHCP options), followed by the 'sname' field.

The values to be passed in an 'option' tag may be too long to fit in the 255 octets available to a single option (e.g., a list of routers in a 'router' option [21]). Options may appear only once, unless otherwise specified in the options document. The client concatenates the values of multiple instances of the same option into a single parameter list for configuration.

DHCP clients are responsible for all message retransmission. The client MUST adopt a retransmission strategy that incorporates a randomized exponential backoff algorithm to determine the delay between retransmissions. The delay between retransmissions SHOULD be chosen to allow sufficient time for replies from the server to be delivered based on the characteristics of the internetwork between the client and the server. For example, in a 10Mb/sec Ethernet internetwork, the delay before the first retransmission SHOULD be 4 seconds randomized by the value of a uniform random number chosen from the range -1 to +1. Clients with clocks that provide resolution granularity of less than one second may choose a non-integer randomization value. The delay before the next retransmission SHOULD be 8 seconds randomized by the value of a uniform number chosen from the range -1 to +1. The retransmission delay SHOULD be doubled with subsequent retransmissions up to a maximum of 64 seconds. The client MAY provide an indication of retransmission attempts to the user as an indication of the progress of the configuration process.

The 'xid' field is used by the client to match incoming DHCP messages with pending requests. A DHCP client MUST choose 'xid's in such a way as to minimize the chance of using an 'xid' identical to one used by another client. For example, a client may choose a different, random initial 'xid' each time the client is rebooted, and subsequently use sequential 'xid's until the next reboot. Selecting a new 'xid' for each retransmission is an implementation decision. A client may choose to reuse the same 'xid' or select a new 'xid' for each retransmitted message.

Normally, DHCP servers and BOOTP relay agents attempt to deliver DHCP OFFER, DHCP ACK and DHCP NAK messages directly to the client using unicast delivery. The IP destination address (in the IP header) is set to the DHCP 'yiaddr' address and the link-layer destination address is set to the DHCP 'chaddr' address. Unfortunately, some client implementations are unable to receive such unicast IP datagrams until the implementation has been configured with a valid IP address (leading to a deadlock in which the client's IP address cannot be delivered until the client has been configured with an IP address).

A client that cannot receive unicast IP datagrams until its protocol software has been configured with an IP address SHOULD set the BROADCAST bit in the 'flags' field to 1 in any DHCP DISCOVER or DHCP REQUEST messages that client sends. The BROADCAST bit will provide a hint to the DHCP server and BOOTP relay agent to broadcast any messages to the client on the client's subnet. A client that can receive unicast IP datagrams before its protocol software has been configured SHOULD clear the BROADCAST bit to 0. The BOOTP clarifications document discusses the ramifications of the use of the BROADCAST bit [21].

A server or relay agent sending or relaying a DHCP message directly to a DHCP client (i.e., not to a relay agent specified in the 'giaddr' field) SHOULD examine the BROADCAST bit in the 'flags' field. If this bit is set to 1, the DHCP message SHOULD be sent as an IP broadcast using an IP broadcast address (preferably 0xffffffff) as the IP destination address and the link-layer broadcast address as the link-layer destination address. If the BROADCAST bit is cleared to 0, the message SHOULD be sent as an IP unicast to the IP address specified in the 'yiaddr' field and the link-layer address specified in the 'chaddr' field. If unicasting is not possible, the message MAY be sent as an IP broadcast using an IP broadcast address (preferably 0xffffffff) as the IP destination address and the link-layer broadcast address as the link-layer destination address.

4.2 DHCP server administrative controls

DHCP servers are not required to respond to every DHCP DISCOVER and DHCP REQUEST message they receive. For example, a network administrator, to retain stringent control over the clients attached to the network, may choose to configure DHCP servers to respond only to clients that have been previously registered through some external mechanism. The DHCP specification describes only the interactions between clients and servers when the clients and servers choose to interact; it is beyond the scope of the DHCP specification to describe all of the administrative controls that system administrators might want to use. Specific DHCP server

implementations may incorporate any controls or policies desired by a network administrator.

In some environments, a DHCP server will have to consider the values of the vendor class options included in DHCPDISCOVER or DHCPREQUEST messages when determining the correct parameters for a particular client.

A DHCP server needs to use some unique identifier to associate a client with its lease. The client MAY choose to explicitly provide the identifier through the 'client identifier' option. If the client supplies a 'client identifier', the client MUST use the same 'client identifier' in all subsequent messages, and the server MUST use that identifier to identify the client. If the client does not provide a 'client identifier' option, the server MUST use the contents of the 'chaddr' field to identify the client. It is crucial for a DHCP client to use an identifier unique within the subnet to which the client is attached in the 'client identifier' option. Use of 'chaddr' as the client's unique identifier may cause unexpected results, as that identifier may be associated with a hardware interface that could be moved to a new client. Some sites may choose to use a manufacturer's serial number as the 'client identifier', to avoid unexpected changes in a clients network address due to transfer of hardware interfaces among computers. Sites may also choose to use a DNS name as the 'client identifier', causing address leases to be associated with the DNS name rather than a specific hardware box.

DHCP clients are free to use any strategy in selecting a DHCP server among those from which the client receives a DHCP OFFER message. The client implementation of DHCP SHOULD provide a mechanism for the user to select directly the 'vendor class identifier' values.

4.3 DHCP server behavior

A DHCP server processes incoming DHCP messages from a client based on the current state of the binding for that client. A DHCP server can receive the following messages from a client:

- o DHCPDISCOVER
- o DHCPREQUEST
- o DHCPDECLINE
- o DHCPRELEASE
- o DHCPINFORM

Table 3 gives the use of the fields and options in a DHCP message by a server. The remainder of this section describes the action of the DHCP server for each possible incoming message.

4.3.1 DHCPDISCOVER message

When a server receives a DHCPDISCOVER message from a client, the server chooses a network address for the requesting client. If no address is available, the server may choose to report the problem to the system administrator. If an address is available, the new address SHOULD be chosen as follows:

- o The client's current address as recorded in the client's current binding, ELSE
- o The client's previous address as recorded in the client's (now expired or released) binding, if that address is in the server's pool of available addresses and not already allocated, ELSE
- o The address requested in the 'Requested IP Address' option, if that address is valid and not already allocated, ELSE
- o A new address allocated from the server's pool of available addresses; the address is selected based on the subnet from which the message was received (if 'giaddr' is 0) or on the address of the relay agent that forwarded the message ('giaddr' when not 0).

As described in section 4.2, a server MAY, for administrative reasons, assign an address other than the one requested, or may refuse to allocate an address to a particular client even though free addresses are available.

Note that, in some network architectures (e.g., internets with more than one IP subnet assigned to a physical network segment), it may be the case that the DHCP client should be assigned an address from a different subnet than the address recorded in 'giaddr'. Thus, DHCP does not require that the client be assigned an address from the subnet in 'giaddr'. A server is free to choose some other subnet, and it is beyond the scope of the DHCP specification to describe ways in which the assigned IP address might be chosen.

While not required for correct operation of DHCP, the server SHOULD NOT reuse the selected network address before the client responds to the server's DHCPOFFER message. The server may choose to record the address as offered to the client.

The server must also choose an expiration time for the lease, as follows:

- o IF the client has not requested a specific lease in the DHCPDISCOVER message and the client already has an assigned network address, the server returns the lease expiration time previously assigned to that address (note that the client must explicitly request a specific lease to extend the expiration time on a previously assigned address), ELSE
- o IF the client has not requested a specific lease in the DHCPDISCOVER message and the client does not have an assigned network address, the server assigns a locally configured default lease time, ELSE
- o IF the client has requested a specific lease in the DHCPDISCOVER message (regardless of whether the client has an assigned network address), the server may choose either to return the requested lease (if the lease is acceptable to local policy) or select another lease.

Field	DHCPOFFER	DHCPACK	DHCNACK
-----	-----	-----	-----
'op'	BOOTREPLY	BOOTREPLY	BOOTREPLY
'htype'	(From "Assigned Numbers" RFC)		
'hlen'	(Hardware address length in octets)		
'hops'	0	0	0
'xid'	'xid' from client	'xid' from client	'xid' from client
	DHCPDISCOVER	DHCPREQUEST	DHCPREQUEST
	message	message	message
'secs'	0	0	0
'ciaddr'	0	'ciaddr' from	0
		DHCPREQUEST or 0	
'yiaddr'	IP address offered	IP address	0
	to client	assigned to client	
'siaddr'	IP address of next	IP address of next	0
	bootstrap server	bootstrap server	
'flags'	'flags' from	'flags' from	'flags' from
	client DHCPDISCOVER	client DHCPREQUEST	client DHCPREQUEST
	message	message	message
'giaddr'	'giaddr' from	'giaddr' from	'giaddr' from
	client DHCPDISCOVER	client DHCPREQUEST	client DHCPREQUEST
	message	message	message
'chaddr'	'chaddr' from	'chaddr' from	'chaddr' from
	client DHCPDISCOVER	client DHCPREQUEST	client DHCPREQUEST
	message	message	message
'sname'	Server host name	Server host name	(unused)
	or options	or options	
'file'	Client boot file	Client boot file	(unused)
	name or options	name or options	
'options'	options	options	

Option	DHCPOFFER	DHCPACK	DHCPNAK
-----	-----	-----	-----
Requested IP address	MUST NOT	MUST NOT	MUST NOT
IP address lease time	MUST	MUST (DHCPREQUEST) MUST NOT (DHCPINFORM)	MUST NOT
Use 'file'/'sname' fields	MAY	MAY	MUST NOT
DHCP message type	DHCPOFFER	DHCPACK	DHCPNAK
Parameter request list	MUST NOT	MUST NOT	MUST NOT
Message	SHOULD	SHOULD	SHOULD
Client identifier	MUST NOT	MUST NOT	MAY
Vendor class identifier	MAY	MAY	MAY
Server identifier	MUST	MUST	MUST
Maximum message size	MUST NOT	MUST NOT	MUST NOT
All others	MAY	MAY	MUST NOT

Table 3: Fields and options used by DHCP servers

Once the network address and lease have been determined, the server constructs a DHCPOFFER message with the offered configuration parameters. It is important for all DHCP servers to return the same parameters (with the possible exception of a newly allocated network address) to ensure predictable client behavior regardless of which server the client selects. The configuration parameters MUST be selected by applying the following rules in the order given below. The network administrator is responsible for configuring multiple DHCP servers to ensure uniform responses from those servers. The server MUST return to the client:

- o The client's network address, as determined by the rules given earlier in this section,
- o The expiration time for the client's lease, as determined by the rules given earlier in this section,
- o Parameters requested by the client, according to the following rules:
 - IF the server has been explicitly configured with a default value for the parameter, the server MUST include that value in an appropriate option in the 'option' field, ELSE
 - IF the server recognizes the parameter as a parameter defined in the Host Requirements Document, the server MUST include the default value for that parameter as given in the Host Requirements Document in an appropriate option in the 'option' field, ELSE
 - The server MUST NOT return a value for that parameter,

The server MUST supply as many of the requested parameters as possible and MUST omit any parameters it cannot provide. The server MUST include each requested parameter only once unless explicitly allowed in the DHCP Options and BOOTP Vendor Extensions document.

- o Any parameters from the existing binding that differ from the Host Requirements Document defaults,
- o Any parameters specific to this client (as identified by the contents of 'chaddr' or 'client identifier' in the DHCPDISCOVER or DHCPREQUEST message), e.g., as configured by the network administrator,
- o Any parameters specific to this client's class (as identified by the contents of the 'vendor class identifier' option in the DHCPDISCOVER or DHCPREQUEST message), e.g., as configured by the network administrator; the parameters MUST be identified by an exact match between the client's vendor class identifiers and the client's classes identified in the server,
- o Parameters with non-default values on the client's subnet.

The server MAY choose to return the 'vendor class identifier' used to determine the parameters in the DHCPOFFER message to assist the client in selecting which DHCPOFFER to accept. The server inserts the 'xid' field from the DHCPDISCOVER message into the 'xid' field of the DHCPOFFER message and sends the DHCPOFFER message to the requesting client.

4.3.2 DHCPREQUEST message

A DHCPREQUEST message may come from a client responding to a DHCPOFFER message from a server, from a client verifying a previously allocated IP address or from a client extending the lease on a network address. If the DHCPREQUEST message contains a 'server identifier' option, the message is in response to a DHCPOFFER message. Otherwise, the message is a request to verify or extend an existing lease. If the client uses a 'client identifier' in a DHCPREQUEST message, it MUST use that same 'client identifier' in all subsequent messages. If the client included a list of requested parameters in a DHCPDISCOVER message, it MUST include that list in all subsequent messages.

Any configuration parameters in the DHCPACK message SHOULD NOT conflict with those in the earlier DHCPOFFER message to which the client is responding. The client SHOULD use the parameters in the DHCPACK message for configuration.

Clients send DHCPREQUEST messages as follows:

- o DHCPREQUEST generated during SELECTING state:

Client inserts the address of the selected server in 'server identifier', 'ciaddr' MUST be zero, 'requested IP address' MUST be filled in with the yiaddr value from the chosen DHCPOFFER.

Note that the client may choose to collect several DHCPOFFER messages and select the "best" offer. The client indicates its selection by identifying the offering server in the DHCPREQUEST message. If the client receives no acceptable offers, the client may choose to try another DHCPDISCOVER message. Therefore, the servers may not receive a specific DHCPREQUEST from which they can decide whether or not the client has accepted the offer. Because the servers have not committed any network address assignments on the basis of a DHCPOFFER, servers are free to reuse offered network addresses in response to subsequent requests. As an implementation detail, servers SHOULD NOT reuse offered addresses and may use an implementation-specific timeout mechanism to decide when to reuse an offered address.

- o DHCPREQUEST generated during INIT-REBOOT state:

'server identifier' MUST NOT be filled in, 'requested IP address' option MUST be filled in with client's notion of its previously assigned address. 'ciaddr' MUST be zero. The client is seeking to verify a previously allocated, cached configuration. Server SHOULD send a DHCPNAK message to the client if the 'requested IP address' is incorrect, or is on the wrong network.

Determining whether a client in the INIT-REBOOT state is on the correct network is done by examining the contents of 'giaddr', the 'requested IP address' option, and a database lookup. If the DHCP server detects that the client is on the wrong net (i.e., the result of applying the local subnet mask or remote subnet mask (if 'giaddr' is not zero) to 'requested IP address' option value doesn't match reality), then the server SHOULD send a DHCPNAK message to the client.

If the network is correct, then the DHCP server should check if the client's notion of its IP address is correct. If not, then the server SHOULD send a DHCPNAK message to the client. If the DHCP server has no record of this client, then it MUST remain silent, and MAY output a warning to the network administrator. This behavior is necessary for peaceful coexistence of non-communicating DHCP servers on the same wire.

If 'giaddr' is 0x0 in the DHCPREQUEST message, the client is on the same subnet as the server. The server MUST broadcast the DHCPNAK message to the 0xffffffff broadcast address because the client may not have a correct network address or subnet mask, and the client may not be answering ARP requests.

If 'giaddr' is set in the DHCPREQUEST message, the client is on a different subnet. The server MUST set the broadcast bit in the DHCPNAK, so that the relay agent will broadcast the DHCPNAK to the client, because the client may not have a correct network address or subnet mask, and the client may not be answering ARP requests.

o DHCPREQUEST generated during RENEWING state:

'server identifier' MUST NOT be filled in, 'requested IP address' option MUST NOT be filled in, 'ciaddr' MUST be filled in with client's IP address. In this situation, the client is completely configured, and is trying to extend its lease. This message will be unicast, so no relay agents will be involved in its transmission. Because 'giaddr' is therefore not filled in, the DHCP server will trust the value in 'ciaddr', and use it when replying to the client.

A client MAY choose to renew or extend its lease prior to T1. The server may choose not to extend the lease (as a policy decision by the network administrator), but should return a DHCPACK message regardless.

o DHCPREQUEST generated during REBINDING state:

'server identifier' MUST NOT be filled in, 'requested IP address' option MUST NOT be filled in, 'ciaddr' MUST be filled in with client's IP address. In this situation, the client is completely configured, and is trying to extend its lease. This message MUST be broadcast to the 0xffffffff IP broadcast address. The DHCP server SHOULD check 'ciaddr' for correctness before replying to the DHCPREQUEST.

The DHCPREQUEST from a REBINDING client is intended to accommodate sites that have multiple DHCP servers and a mechanism for maintaining consistency among leases managed by multiple servers. A DHCP server MAY extend a client's lease only if it has local administrative authority to do so.

4.3.3 DHCPDECLINE message

If the server receives a DHCPDECLINE message, the client has discovered through some other means that the suggested network address is already in use. The server MUST mark the network address as not available and SHOULD notify the local system administrator of a possible configuration problem.

4.3.4 DHCPRELEASE message

Upon receipt of a DHCPRELEASE message, the server marks the network address as not allocated. The server SHOULD retain a record of the client's initialization parameters for possible reuse in response to subsequent requests from the client.

4.3.5 DHCPINFORM message

The server responds to a DHCPINFORM message by sending a DHCPACK message directly to the address given in the 'ciaddr' field of the DHCPINFORM message. The server MUST NOT send a lease expiration time to the client and SHOULD NOT fill in 'yiaddr'. The server includes other parameters in the DHCPACK message as defined in section 4.3.1.

4.3.6 Client messages

Table 4 details the differences between messages from clients in various states.

	INIT-REBOOT	SELECTING	RENEWING	REBINDING
broad/unicast	broadcast	broadcast	unicast	broadcast
server-ip	MUST NOT	MUST	MUST NOT	MUST NOT
requested-ip	MUST	MUST	MUST NOT	MUST NOT
ciaddr	zero	zero	IP address	IP address

Table 4: Client messages from different states

4.4 DHCP client behavior

Figure 5 gives a state-transition diagram for a DHCP client. A client can receive the following messages from a server:

- o DHCPOFFER
- o DHCPACK
- o DHCPNAK

The DHCPINFORM message is not shown in figure 5. A client simply sends the DHCPINFORM and waits for DHCPACK messages. Once the client has selected its parameters, it has completed the configuration process.

Table 5 gives the use of the fields and options in a DHCP message by a client. The remainder of this section describes the action of the DHCP client for each possible incoming message. The description in the following section corresponds to the full configuration procedure previously described in section 3.1, and the text in the subsequent section corresponds to the abbreviated configuration procedure described in section 3.2.

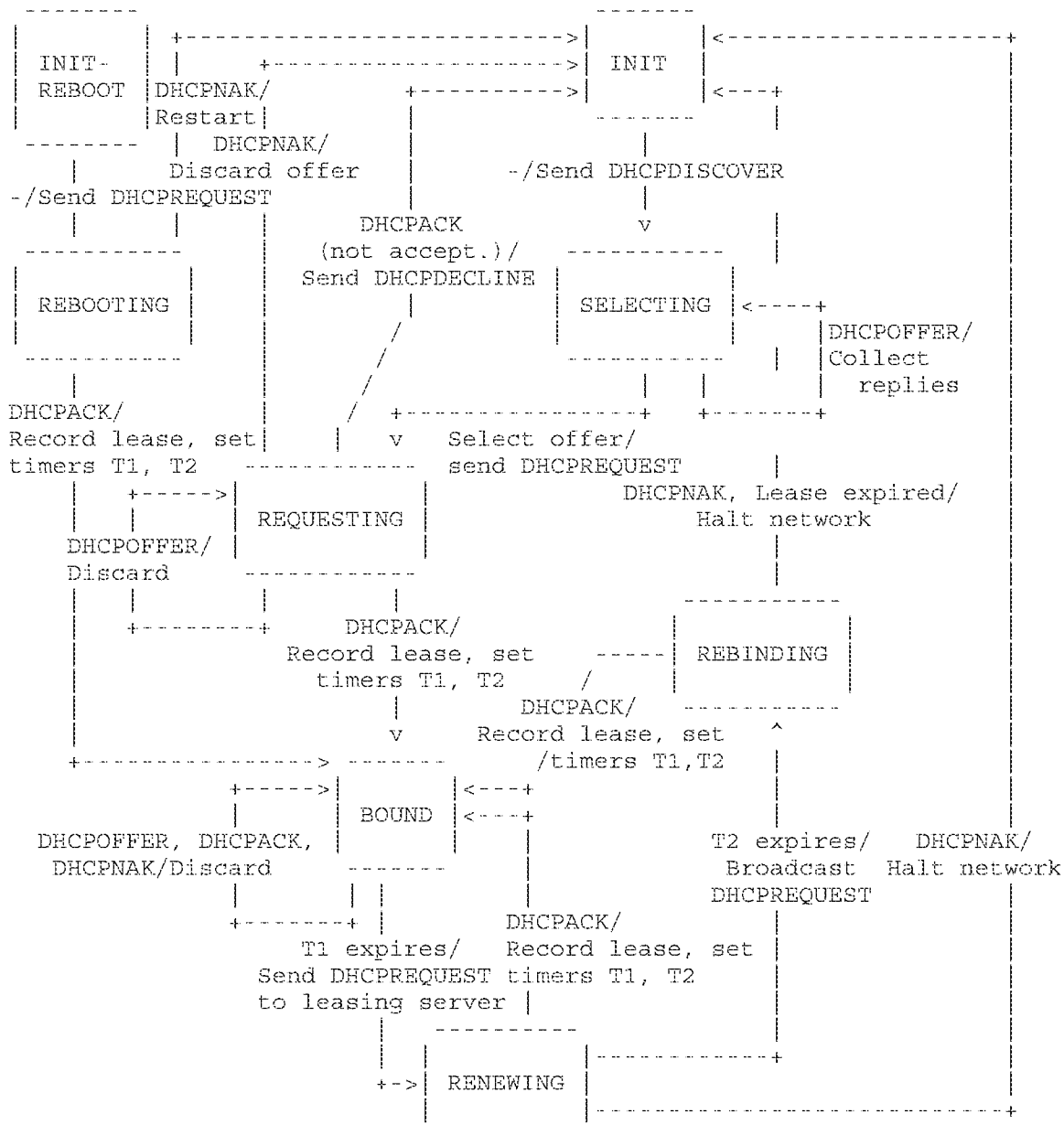


Figure 5: State-transition diagram for DHCP clients

4.4.1 Initialization and allocation of network address

The client begins in INIT state and forms a DHCPDISCOVER message. The client SHOULD wait a random time between one and ten seconds to desynchronize the use of DHCP at startup. The client sets 'ciaddr' to 0x00000000. The client MAY request specific parameters by including the 'parameter request list' option. The client MAY suggest a network address and/or lease time by including the 'requested IP address' and 'IP address lease time' options. The client MUST include its hardware address in the 'chaddr' field, if necessary for delivery of DHCP reply messages. The client MAY include a different unique identifier in the 'client identifier' option, as discussed in section 4.2. If the client included a list of requested parameters in a DHCPDISCOVER message, it MUST include that list in all subsequent messages.

The client generates and records a random transaction identifier and inserts that identifier into the 'xid' field. The client records its own local time for later use in computing the lease expiration. The client then broadcasts the DHCPDISCOVER on the local hardware broadcast address to the 0xffffffff IP broadcast address and 'DHCP server' UDP port.

If the 'xid' of an arriving DHCPOFFER message does not match the 'xid' of the most recent DHCPDISCOVER message, the DHCPOFFER message must be silently discarded. Any arriving DHCPACK messages must be silently discarded.

The client collects DHCPOFFER messages over a period of time, selects one DHCPOFFER message from the (possibly many) incoming DHCPOFFER messages (e.g., the first DHCPOFFER message or the DHCPOFFER message from the previously used server) and extracts the server address from the 'server identifier' option in the DHCPOFFER message. The time over which the client collects messages and the mechanism used to select one DHCPOFFER are implementation dependent.

Field	DHCPDISCOVER DHCPIFORM	DHCPREQUEST DHCPREQUEST	DHCPDECLINE, DHCPRELEASE
-----	-----	-----	-----
'op'	BOOTREQUEST	BOOTREQUEST	BOOTREQUEST
'htype'	(From "Assigned Numbers" RFC)		
'hlen'	(Hardware address length in octets)		
'hops'	0	0	0
'xid'	selected by client	'xid' from server DHCP OFFER message	selected by client
'secs'	0 or seconds since DHCP process started	0 or seconds since DHCP process started	0
'flags'	Set 'BROADCAST' flag if client requires broadcast reply	Set 'BROADCAST' flag if client requires broadcast reply	0
'ciaddr'	0 (DHCPDISCOVER) client's network address (DHCPIFORM)	0 or client's network address (BOUND/RENEW/REBIND)	0 (DHCPDECLINE) client's network address (DHCPRELEASE)
'yiaddr'	0	0	0
'siaddr'	0	0	0
'giaddr'	0	0	0
'chaddr'	client's hardware address	client's hardware address	client's hardware address
'sname'	options, if indicated in 'sname/file' option; otherwise unused	options, if indicated in 'sname/file' option; otherwise unused	(unused)
'file'	options, if indicated in 'sname/file' option; otherwise unused	options, if indicated in 'sname/file' option; otherwise unused	(unused)
'options'	options	options	(unused)

Option	DHCPDISCOVER DHCPIFORM	DHCPREQUEST	DHCPDECLINE, DHCPRELEASE
-----	-----	-----	-----
Requested IP address	MAY (DISCOVER) MUST NOT (INFORM)	MUST (in SELECTING or INIT-REBOOT) MUST NOT (in BOUND or RENEWING)	MUST (DHCPDECLINE), MUST NOT (DHCPRELEASE)
IP address lease time	MAY (DISCOVER) MUST NOT (INFORM)	MAY	MUST NOT
Use 'file'/'sname' fields	MAY	MAY	MAY
DHCP message type	DHCPDISCOVER/ DHCPIFORM	DHCPREQUEST	DHCPDECLINE/ DHCPRELEASE
Client identifier	MAY	MAY	MAY
Vendor class identifier	MAY	MAY	MUST NOT
Server identifier	MUST NOT	MUST (after SELECTING) MUST NOT (after INIT-REBOOT, BOUND, RENEWING or REBINDING)	MUST
Parameter request list	MAY	MAY	MUST NOT
Maximum message size	MAY	MAY	MUST NOT
Message	SHOULD NOT	SHOULD NOT	SHOULD
Site-specific	MAY	MAY	MUST NOT
All others	MAY	MAY	MUST NOT

Table 5: Fields and options used by DHCP clients

If the parameters are acceptable, the client records the address of the server that supplied the parameters from the 'server identifier' field and sends that address in the 'server identifier' field of a DHCPREQUEST broadcast message. Once the DHCPACK message from the server arrives, the client is initialized and moves to BOUND state. The DHCPREQUEST message contains the same 'xid' as the DHCPDISCOVER message. The client records the lease expiration time as the sum of the time at which the original request was sent and the duration of the lease from the DHCPACK message. The client SHOULD perform a check on the suggested address to ensure that the address is not already in use. For example, if the client is on a network that supports ARP, the client may issue an ARP request for the suggested request. When broadcasting an ARP request for the suggested address, the client must fill in its own hardware address as the sender's hardware address, and 0 as the sender's IP address, to avoid confusing ARP caches in other hosts on the same subnet. If the

network address appears to be in use, the client MUST send a DHCPDECLINE message to the server. The client SHOULD broadcast an ARP reply to announce the client's new IP address and clear any outdated ARP cache entries in hosts on the client's subnet.

4.4.2 Initialization with known network address

The client begins in INIT-REBOOT state and sends a DHCPREQUEST message. The client MUST insert its known network address as a 'requested IP address' option in the DHCPREQUEST message. The client may request specific configuration parameters by including the 'parameter request list' option. The client generates and records a random transaction identifier and inserts that identifier into the 'xid' field. The client records its own local time for later use in computing the lease expiration. The client MUST NOT include a 'server identifier' in the DHCPREQUEST message. The client then broadcasts the DHCPREQUEST on the local hardware broadcast address to the 'DHCP server' UDP port.

Once a DHCPACK message with an 'xid' field matching that in the client's DHCPREQUEST message arrives from any server, the client is initialized and moves to BOUND state. The client records the lease expiration time as the sum of the time at which the DHCPREQUEST message was sent and the duration of the lease from the DHCPACK message.

4.4.3 Initialization with an externally assigned network address

The client sends a DHCPINFORM message. The client may request specific configuration parameters by including the 'parameter request list' option. The client generates and records a random transaction identifier and inserts that identifier into the 'xid' field. The client places its own network address in the 'ciaddr' field. The client SHOULD NOT request lease time parameters.

The client then unicasts the DHCPINFORM to the DHCP server if it knows the server's address, otherwise it broadcasts the message to the limited (all 1s) broadcast address. DHCPINFORM messages MUST be directed to the 'DHCP server' UDP port.

Once a DHCPACK message with an 'xid' field matching that in the client's DHCPINFORM message arrives from any server, the client is initialized.

If the client does not receive a DHCPACK within a reasonable period of time (60 seconds or 4 tries if using timeout suggested in section 4.1), then it SHOULD display a message informing the user of the problem, and then SHOULD begin network processing using suitable

defaults as per Appendix A.

4.4.4 Use of broadcast and unicast

The DHCP client broadcasts DHCPDISCOVER, DHCPREQUEST and DHCPINFORM messages, unless the client knows the address of a DHCP server. The client unicasts DHCPRELEASE messages to the server. Because the client is declining the use of the IP address supplied by the server, the client broadcasts DHCPDECLINE messages.

When the DHCP client knows the address of a DHCP server, in either INIT or REBOOTING state, the client may use that address in the DHCPDISCOVER or DHCPREQUEST rather than the IP broadcast address. The client may also use unicast to send DHCPINFORM messages to a known DHCP server. If the client receives no response to DHCP messages sent to the IP address of a known DHCP server, the DHCP client reverts to using the IP broadcast address.

4.4.5 Reacquisition and expiration

The client maintains two times, T1 and T2, that specify the times at which the client tries to extend its lease on its network address. T1 is the time at which the client enters the RENEWING state and attempts to contact the server that originally issued the client's network address. T2 is the time at which the client enters the REBINDING state and attempts to contact any server. T1 MUST be earlier than T2, which, in turn, MUST be earlier than the time at which the client's lease will expire.

To avoid the need for synchronized clocks, T1 and T2 are expressed in options as relative times [2].

At time T1 the client moves to RENEWING state and sends (via unicast) a DHCPREQUEST message to the server to extend its lease. The client sets the 'ciaddr' field in the DHCPREQUEST to its current network address. The client records the local time at which the DHCPREQUEST message is sent for computation of the lease expiration time. The client MUST NOT include a 'server identifier' in the DHCPREQUEST message.

Any DHCPACK messages that arrive with an 'xid' that does not match the 'xid' of the client's DHCPREQUEST message are silently discarded. When the client receives a DHCPACK from the server, the client computes the lease expiration time as the sum of the time at which the client sent the DHCPREQUEST message and the duration of the lease in the DHCPACK message. The client has successfully reacquired its network address, returns to BOUND state and may continue network processing.

If no DHCPACK arrives before time T2, the client moves to REBINDING state and sends (via broadcast) a DHCPREQUEST message to extend its lease. The client sets the 'ciaddr' field in the DHCPREQUEST to its current network address. The client MUST NOT include a 'server identifier' in the DHCPREQUEST message.

Times T1 and T2 are configurable by the server through options. T1 defaults to $(0.5 * \text{duration_of_lease})$. T2 defaults to $(0.875 * \text{duration_of_lease})$. Times T1 and T2 SHOULD be chosen with some random "fuzz" around a fixed value, to avoid synchronization of client reacquisition.

A client MAY choose to renew or extend its lease prior to T1. The server MAY choose to extend the client's lease according to policy set by the network administrator. The server SHOULD return T1 and T2, and their values SHOULD be adjusted from their original values to take account of the time remaining on the lease.

In both RENEWING and REBINDING states, if the client receives no response to its DHCPREQUEST message, the client SHOULD wait one-half of the remaining time until T2 (in RENEWING state) and one-half of the remaining lease time (in REBINDING state), down to a minimum of 60 seconds, before retransmitting the DHCPREQUEST message.

If the lease expires before the client receives a DHCPACK, the client moves to INIT state, MUST immediately stop any other network processing and requests network initialization parameters as if the client were uninitialized. If the client then receives a DHCPACK allocating that client its previous network address, the client SHOULD continue network processing. If the client is given a new network address, it MUST NOT continue using the previous network address and SHOULD notify the local users of the problem.

4.4.6 DHCPRELEASE

If the client no longer requires use of its assigned network address (e.g., the client is gracefully shut down), the client sends a DHCPRELEASE message to the server. Note that the correct operation of DHCP does not depend on the transmission of DHCPRELEASE messages.

5. Acknowledgments

The author thanks the many (and too numerous to mention!) members of the DHC WG for their tireless and ongoing efforts in the development of DHCP and this document.

The efforts of J Allard, Mike Carney, Dave Lapp, Fred Lien and John Mendonca in organizing DHCP interoperability testing sessions are gratefully acknowledged.

The development of this document was supported in part by grants from the Corporation for National Research Initiatives (CNRI), Bucknell University and Sun Microsystems.

6. References

- [1] Acetta, M., "Resource Location Protocol", RFC 887, CMU, December 1983.
- [2] Alexander, S., and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 1533, Lachman Technology, Inc., Bucknell University, October 1993.
- [3] Braden, R., Editor, "Requirements for Internet Hosts -- Communication Layers", STD 3, RFC 1122, USC/Information Sciences Institute, October 1989.
- [4] Braden, R., Editor, "Requirements for Internet Hosts -- Application and Support, STD 3, RFC 1123, USC/Information Sciences Institute, October 1989.
- [5] Brownell, D, "Dynamic Reverse Address Resolution Protocol (DRARP)", Work in Progress.
- [6] Comer, D., and R. Droms, "Uniform Access to Internet Directory Services", Proc. of ACM SIGCOMM '90 (Special issue of Computer Communications Review), 20(4):50--59, 1990.
- [7] Croft, B., and J. Gilmore, "Bootstrap Protocol (BOOTP)", RFC 951, Stanford and SUN Microsystems, September 1985.
- [8] Deering, S., "ICMP Router Discovery Messages", RFC 1256, Xerox PARC, September 1991.
- [9] Droms, D., "Interoperation between DHCP and BOOTP", RFC 1534, Bucknell University, October 1993.

- [10] Finlayson, R., Mann, T., Mogul, J., and M. Theimer, "A Reverse Address Resolution Protocol", RFC 903, Stanford, June 1984.
- [11] Gray C., and D. Cheriton, "Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency", In Proc. of the Twelfth ACM Symposium on Operating Systems Design, 1989.
- [12] Mockapetris, P., "Domain Names -- Concepts and Facilities", STD 13, RFC 1034, USC/Information Sciences Institute, November 1987.
- [13] Mockapetris, P., "Domain Names -- Implementation and Specification", STD 13, RFC 1035, USC/Information Sciences Institute, November 1987.
- [14] Mogul J., and S. Deering, "Path MTU Discovery", RFC 1191, November 1990.
- [15] Morgan, R., "Dynamic IP Address Assignment for Ethernet Attached Hosts", Work in Progress.
- [16] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, USC/Information Sciences Institute, September 1981.
- [17] Reynolds, J., "BOOTP Vendor Information Extensions", RFC 1497, USC/Information Sciences Institute, August 1993.
- [18] Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC 1700, USC/Information Sciences Institute, October 1994.
- [19] Jeffrey Schiller and Mark Rosenstein. A Protocol for the Dynamic Assignment of IP Addresses for use on an Ethernet. (Available from the Athena Project, MIT), 1989.
- [20] Sollins, K., "The TFTP Protocol (Revision 2)", RFC 783, NIC, June 1981.
- [21] Wimer, W., "Clarifications and Extensions for the Bootstrap Protocol", RFC 1542, Carnegie Mellon University, October 1993.

7. Security Considerations

DHCP is built directly on UDP and IP which are as yet inherently insecure. Furthermore, DHCP is generally intended to make maintenance of remote and/or diskless hosts easier. While perhaps not impossible, configuring such hosts with passwords or keys may be difficult and inconvenient. Therefore, DHCP in its current form is quite insecure.

Unauthorized DHCP servers may be easily set up. Such servers can then send false and potentially disruptive information to clients such as incorrect or duplicate IP addresses, incorrect routing information (including spoof routers, etc.), incorrect domain nameserver addresses (such as spoof nameservers), and so on. Clearly, once this seed information is in place, an attacker can further compromise affected systems.

Malicious DHCP clients could masquerade as legitimate clients and retrieve information intended for those legitimate clients. Where dynamic allocation of resources is used, a malicious client could claim all resources for itself, thereby denying resources to legitimate clients.

8. Author's Address

Ralph Droms
Computer Science Department
323 Dana Engineering
Bucknell University
Lewisburg, PA 17837

Phone: (717) 524-1145
EMail: droms@bucknell.edu

A. Host Configuration Parameters

IP-layer_parameters,_per_host:__

Be a router	on/off	HRC 3.1
Non-local source routing	on/off	HRC 3.3.5
Policy filters for non-local source routing	(list)	HRC 3.3.5
Maximum reassembly size	integer	HRC 3.3.2
Default TTL	integer	HRC 3.2.1.7
PMTU aging timeout	integer	MTU 6.6
MTU plateau table	(list)	MTU 7
IP-layer_parameters,_per_interface:__		
IP address	(address)	HRC 3.3.1.6
Subnet mask	(address mask)	HRC 3.3.1.6
MTU	integer	HRC 3.3.3
All-subnets-MTU	on/off	HRC 3.3.3
Broadcast address flavor	0x00000000/0xffffffff	HRC 3.3.6
Perform mask discovery	on/off	HRC 3.2.2.9
Be a mask supplier	on/off	HRC 3.2.2.9
Perform router discovery	on/off	RD 5.1
Router solicitation address	(address)	RD 5.1
Default routers, list of:		
router address	(address)	HRC 3.3.1.6
preference level	integer	HRC 3.3.1.6
Static routes, list of:		
destination	(host/subnet/net)	HRC 3.3.1.2
destination mask	(address mask)	HRC 3.3.1.2
type-of-service	integer	HRC 3.3.1.2
first-hop router	(address)	HRC 3.3.1.2
ignore redirects	on/off	HRC 3.3.1.2
PMTU	integer	MTU 6.6
perform PMTU discovery	on/off	MTU 6.6

Link-layer_parameters,_per_interface:__

Trailers	on/off	HRC 2.3.1
ARP cache timeout	integer	HRC 2.3.2.1
Ethernet encapsulation	(RFC 894/RFC 1042)	HRC 2.3.3

TCP_parameters,_per_host:__

TTL	integer	HRC 4.2.2.19
Keep-alive interval	integer	HRC 4.2.3.6
Keep-alive data size	0/1	HRC 4.2.3.6

Key:

MTU = Path MTU Discovery (RFC 1191, Proposed Standard)

RD = Router Discovery (RFC 1256, Proposed Standard)

Network Working Group
Request for Comments: 1533
Obsoletes: 1497, 1395, 1084, 1048
Category: Standards Track

S. Alexander
Lachman Technology, Inc.
R. Droms
Bucknell University
October 1993

DHCP Options and BOOTP Vendor Extensions

Status of this Memo

This RFC specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

The Dynamic Host Configuration Protocol (DHCP) [1] provides a framework for passing configuration information to hosts on a TCP/IP network. Configuration parameters and other control information are carried in tagged data items that are stored in the "options" field of the DHCP message. The data items themselves are also called "options."

This document specifies the current set of DHCP options. This document will be periodically updated as new options are defined. Each superseding document will include the entire current list of valid options.

All of the vendor information extensions defined in RFC 1497 [2] may be used as DHCP options. The definitions given in RFC 1497 are included in this document, which supersedes RFC 1497. All of the DHCP options defined in this document, except for those specific to DHCP as defined in section 9, may be used as BOOTP vendor information extensions.

Table of Contents

1. Introduction	2
2. BOOTP Extension/DHCP Option Field Format	2
3. RFC 1497 Vendor Extensions	3
4. IP Layer Parameters per Host	10
5. IP Layer Parameters per Interface	13
6. Link Layer Parameters per Interface	16
7. TCP Parameters	17
8. Application and Service Parameters	18

Alexander & Droms

[Page 1]

9. DHCP Extensions	23
10. Extensions	27
11. Acknowledgements	28
12. References	28
13. Security Considerations	19
14. Authors' Addresses	30

1. Introduction

This document specifies options for use with both the Dynamic Host Configuration Protocol and the Bootstrap Protocol.

The full description of DHCP packet formats may be found in the DHCP specification document [1], and the full description of BOOTP packet formats may be found in the BOOTP specification document [3]. This document defines the format of information in the last field of DHCP packets ('options') and of BOOTP packets ('vend'). The remainder of this section defines a generalized use of this area for giving information useful to a wide class of machines, operating systems and configurations. Sites with a single DHCP or BOOTP server that is shared among heterogeneous clients may choose to define other, site-specific formats for the use of the 'options' field.

Section 2 of this memo describes the formats of DHCP options and BOOTP vendor extensions. Section 3 describes options defined in previous documents for use with BOOTP (all may also be used with DHCP). Sections 4-8 define new options intended for use with both DHCP and BOOTP. Section 9 defines options used only in DHCP.

References further describing most of the options defined in sections 2-6 can be found in section 12. The use of the options defined in section 9 is described in the DHCP specification [1].

Information on registering new options is contained in section 10.

2. BOOTP Extension/DHCP Option Field Format

DHCP options have the same format as the BOOTP "vendor extensions" defined in RFC 1497 [2]. Options may be fixed length or variable length. All options begin with a tag octet, which uniquely identifies the option. Fixed-length options without data consist of only a tag octet. Only options 0 and 255 are fixed length. All other options are variable-length with a length octet following the tag octet. The value of the length octet does not include the two octets specifying the tag and length. The length octet is followed by "length" octets of data. In the case of some variable-length options the length field is a constant but must still be specified.

Any options defined subsequent to this document should contain a length octet even if the length is fixed or zero.

All multi-octet quantities are in network byte-order.

When used with BOOTP, the first four octets of the vendor information field have been assigned to the "magic cookie" (as suggested in RFC 951). This field identifies the mode in which the succeeding data is to be interpreted. The value of the magic cookie is the 4 octet dotted decimal 99.130.83.99 (or hexadecimal number 63.82.53.63) in network byte order.

All of the "vendor extensions" defined in RFC 1497 are also DHCP options.

Option codes 128 to 254 (decimal) are reserved for site-specific options.

Except for the options in section 9, all options may be used with either DHCP or BOOTP.

Many of these options have their default values specified in other documents. In particular, RFC 1122 [4] specifies default values for most IP and TCP configuration parameters.

3. RFC 1497 Vendor Extensions

This section lists the vendor extensions as defined in RFC 1497. They are defined here for completeness.

3.1. Pad Option

The pad option can be used to cause subsequent fields to align on word boundaries.

The code for the pad option is 0, and its length is 1 octet.

```
Code
+-----+
|  0  |
+-----+
```

3.2. End Option

The end option marks the end of valid information in the vendor field. Subsequent octets should be filled with pad options.

The code for the end option is 255, and its length is 1 octet.

Code
255

3.3. Subnet Mask

The subnet mask option specifies the client's subnet mask as per RFC 950 [5].

If both the subnet mask and the router option are specified in a DHCP reply, the subnet mask option MUST be first.

The code for the subnet mask option is 1, and its length is 4 octets.

Code	Len	Subnet Mask			
1	4	m1	m2	m3	m4

3.4. Time Offset

The time offset field specifies the offset of the client's subnet in seconds from Coordinated Universal Time (UTC). The offset is expressed as a signed 32-bit integer.

The code for the time offset option is 2, and its length is 4 octets.

Code	Len	Time Offset			
2	4	n1	n2	n3	n4

3.5. Router Option

The router option specifies a list of IP addresses for routers on the client's subnet. Routers SHOULD be listed in order of preference.

The code for the router option is 3. The minimum length for the router option is 4 octets, and the length MUST always be a multiple of 4.

Code	Len	Address 1				Address 2			
3	n	a1	a2	a3	a4	a1	a2	...	

3.6. Time Server Option

The time server option specifies a list of RFC 868 [6] time servers available to the client. Servers SHOULD be listed in order of preference.

The code for the time server option is 4. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

Code	Len	Address 1				Address 2			
4	n	a1	a2	a3	a4	a1	a2	...	

3.7. Name Server Option

The name server option specifies a list of IEN 116 [7] name servers available to the client. Servers SHOULD be listed in order of preference.

The code for the name server option is 5. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

Code	Len	Address 1				Address 2			
5	n	a1	a2	a3	a4	a1	a2	...	

3.8. Domain Name Server Option

The domain name server option specifies a list of Domain Name System (STD 13, RFC 1035 [8]) name servers available to the client. Servers SHOULD be listed in order of preference.

The code for the domain name server option is 6. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

Code	Len	Address 1				Address 2			
6	n	a1	a2	a3	a4	a1	a2	...	

3.9. Log Server Option

The log server option specifies a list of MIT-LCS UDP log servers available to the client. Servers SHOULD be listed in order of preference.

The code for the log server option is 7. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

Code	Len	Address 1				Address 2			
7	n	a1	a2	a3	a4	a1	a2	...	

3.10. Cookie Server Option

The cookie server option specifies a list of RFC 865 [9] cookie servers available to the client. Servers SHOULD be listed in order of preference.

The code for the log server option is 8. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

Code	Len	Address 1				Address 2			
8	n	a1	a2	a3	a4	a1	a2	...	

3.11. LPR Server Option

The LPR server option specifies a list of RFC 1179 [10] line printer servers available to the client. Servers SHOULD be listed in order of preference.

The code for the LPR server option is 9. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

Code	Len	Address 1				Address 2			
9	n	a1	a2	a3	a4	a1	a2	...	

3.12. Impress Server Option

The Impress server option specifies a list of Imagen Impress servers available to the client. Servers SHOULD be listed in order of preference.

The code for the Impress server option is 10. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

Code	Len	Address 1				Address 2			
10	n	a1	a2	a3	a4	a1	a2	...	

3.13. Resource Location Server Option

This option specifies a list of RFC 887 [11] Resource Location servers available to the client. Servers SHOULD be listed in order of preference.

The code for this option is 11. The minimum length for this option is 4 octets, and the length MUST always be a multiple of 4.

Code	Len	Address 1				Address 2			
11	n	a1	a2	a3	a4	a1	a2	...	

3.14. Host Name Option

This option specifies the name of the client. The name may or may not be qualified with the local domain name (see section 3.17 for the preferred way to retrieve the domain name). See RFC 1035 for character set restrictions.

The code for this option is 12, and its minimum length is 1.

Code	Len	Host Name						
12	n	h1	h2	h3	h4	h5	h6	...

3.15. Boot File Size Option

This option specifies the length in 512-octet blocks of the default boot image for the client. The file length is specified as an unsigned 16-bit integer.

The code for this option is 13, and its length is 2.

Code	Len	File Size	
13	2	11	12

3.16. Merit Dump File

This option specifies the path-name of a file to which the client's core image should be dumped in the event the client crashes. The path is formatted as a character string consisting of characters from the NVT ASCII character set.

The code for this option is 14. Its minimum length is 1.

Code	Len	Dump File Pathname				
14	n	n1	n2	n3	n4	...

3.17. Domain Name

This option specifies the domain name that client should use when resolving hostnames via the Domain Name System.

The code for this option is 15. Its minimum length is 1.

Code	Len	Domain Name				
15	n	d1	d2	d3	d4	...

3.18. Swap Server

This specifies the IP address of the client's swap server.

The code for this option is 16 and its length is 4.

Code	Len	Swap Server Address			
16	n	a1	a2	a3	a4

3.19. Root Path

This option specifies the path-name that contains the client's root disk. The path is formatted as a character string consisting of characters from the NVT ASCII character set.

The code for this option is 17. Its minimum length is 1.

Code	Len	Root Disk Pathname				
17	n	n1	n2	n3	n4	...

3.20. Extensions Path

A string to specify a file, retrievable via TFTP, which contains information which can be interpreted in the same way as the 64-octet vendor-extension field within the BOOTP response, with the following exceptions:

- the length of the file is unconstrained;
- all references to Tag 18 (i.e., instances of the BOOTP Extensions Path field) within the file are ignored.

The code for this option is 18. Its minimum length is 1.

Code	Len	Extensions Pathname				
18	n	n1	n2	n3	n4	...

4. IP Layer Parameters per Host

This section details the options that affect the operation of the IP layer on a per-host basis.

4.1. IP Forwarding Enable/Disable Option

This option specifies whether the client should configure its IP layer for packet forwarding. A value of 0 means disable IP forwarding, and a value of 1 means enable IP forwarding.

The code for this option is 19, and its length is 1.

Code	Len	Value
19	1	0/1

4.2. Non-Local Source Routing Enable/Disable Option

This option specifies whether the client should configure its IP layer to allow forwarding of datagrams with non-local source routes (see Section 3.3.5 of [4] for a discussion of this topic). A value of 0 means disallow forwarding of such datagrams, and a value of 1 means allow forwarding.

The code for this option is 20, and its length is 1.

Code	Len	Value
20	1	0/1

4.3. Policy Filter Option

This option specifies policy filters for non-local source routing. The filters consist of a list of IP addresses and masks which specify destination/mask pairs with which to filter incoming source routes.

Any source routed datagram whose next-hop address does not match one of the filters should be discarded by the client.

See [4] for further information.

The code for this option is 21. The minimum length of this option is 8, and the length MUST be a multiple of 8.

Code	Len	Address 1				Mask 1			
21	n	a1	a2	a3	a4	m1	m2	m3	m4

Address 2				Mask 2				
a1	a2	a3	a4	m1	m2	m3	m4	...

4.4. Maximum Datagram Reassembly Size

This option specifies the maximum size datagram that the client should be prepared to reassemble. The size is specified as a 16-bit unsigned integer. The minimum value legal value is 576.

The code for this option is 22, and its length is 2.

Code	Len	Size	
22	2	s1	s2

4.5. Default IP Time-to-live

This option specifies the default time-to-live that the client should use on outgoing datagrams. The TTL is specified as an octet with a value between 1 and 255.

The code for this option is 23, and its length is 1.

Code	Len	TTL
23	1	t1

4.6. Path MTU Aging Timeout Option

This option specifies the timeout (in seconds) to use when aging Path MTU values discovered by the mechanism defined in RFC 1191 [12]. The timeout is specified as a 32-bit unsigned integer.

The code for this option is 24, and its length is 4.

Code	Len	Timeout			
24	4	t1	t2	t3	t4

4.7. Path MTU Plateau Table Option

This option specifies a table of MTU sizes to use when performing Path MTU Discovery as defined in RFC 1191. The table is formatted as a list of 16-bit unsigned integers, ordered from smallest to largest. The minimum MTU value cannot be smaller than 68.

The code for this option is 25. Its minimum length is 2, and the length MUST be a multiple of 2.

Code	Len	Size 1		Size 2	
25	n	s1	s2	s1	s2 ...

5. IP Layer Parameters per Interface

This section details the options that affect the operation of the IP layer on a per-interface basis. It is expected that a client can issue multiple requests, one per interface, in order to configure interfaces with their specific parameters.

5.1. Interface MTU Option

This option specifies the MTU to use on this interface. The MTU is specified as a 16-bit unsigned integer. The minimum legal value for the MTU is 68.

The code for this option is 26, and its length is 2.

Code	Len	MTU	
26	2	m1	m2

5.2. All Subnets are Local Option

This option specifies whether or not the client may assume that all subnets of the IP network to which the client is connected use the same MTU as the subnet of that network to which the client is directly connected. A value of 1 indicates that all subnets share the same MTU. A value of 0 means that the client should assume that some subnets of the directly connected network may have smaller MTUs.

The code for this option is 27, and its length is 1.

Code	Len	Value
27	1	0/1

5.3. Broadcast Address Option

This option specifies the broadcast address in use on the client's subnet. Legal values for broadcast addresses are specified in section 3.2.1.3 of [4].

The code for this option is 28, and its length is 4.

Code	Len	Broadcast Address
28	4	b1 b2 b3 b4

5.4. Perform Mask Discovery Option

This option specifies whether or not the client should perform subnet mask discovery using ICMP. A value of 0 indicates that the client should not perform mask discovery. A value of 1 means that the client should perform mask discovery.

The code for this option is 29, and its length is 1.

Code	Len	Value
29	1	0/1

5.5. Mask Supplier Option

This option specifies whether or not the client should respond to subnet mask requests using ICMP. A value of 0 indicates that the client should not respond. A value of 1 means that the client should respond.

The code for this option is 30, and its length is 1.

Code	Len	Value
30	1	0/1

5.6. Perform Router Discovery Option

This option specifies whether or not the client should solicit routers using the Router Discovery mechanism defined in RFC 1256 [13]. A value of 0 indicates that the client should not perform router discovery. A value of 1 means that the client should perform router discovery.

The code for this option is 31, and its length is 1.

Code	Len	Value
31	1	0/1

5.7. Router Solicitation Address Option

This option specifies the address to which the client should transmit router solicitation requests.

The code for this option is 32, and its length is 4.

Code	Len	Address			
32	4	a1	a2	a3	a4

5.8. Static Route Option

This option specifies a list of static routes that the client should install in its routing cache. If multiple routes to the same destination are specified, they are listed in descending order of priority.

The routes consist of a list of IP address pairs. The first address is the destination address, and the second address is the router for the destination.

The default route (0.0.0.0) is an illegal destination for a static route. See section 3.5 for information about the router option.

The code for this option is 33. The minimum length of this option is 8, and the length MUST be a multiple of 8.

Code	Len	Destination 1				Router 1			
33	n	d1	d2	d3	d4	r1	r2	r3	r4
		Destination 2				Router 2			
		d1	d2	d3	d4	r1	r2	r3	r4 ...

6. Link Layer Parameters per Interface

This section lists the options that affect the operation of the data link layer on a per-interface basis.

6.1. Trailer Encapsulation Option

This option specifies whether or not the client should negotiate the use of trailers (RFC 893 [14]) when using the ARP protocol. A value of 0 indicates that the client should not attempt to use trailers. A value of 1 means that the client should attempt to use trailers.

The code for this option is 34, and its length is 1.

Code	Len	Value
34	1	0/1

6.2. ARP Cache Timeout Option

This option specifies the timeout in seconds for ARP cache entries. The time is specified as a 32-bit unsigned integer.

The code for this option is 35, and its length is 4.

Code	Len	Time			
35	4	t1	t2	t3	t4

6.3. Ethernet Encapsulation Option

This option specifies whether or not the client should use Ethernet Version 2 (RFC 894 [15]) or IEEE 802.3 (RFC 1042 [16]) encapsulation if the interface is an Ethernet. A value of 0 indicates that the client should use RFC 894 encapsulation. A value of 1 means that the client should use RFC 1042 encapsulation.

The code for this option is 36, and its length is 1.

Code	Len	Value
36	1	0/1

7. TCP Parameters

This section lists the options that affect the operation of the TCP layer on a per-interface basis.

7.1. TCP Default TTL Option

This option specifies the default TTL that the client should use when sending TCP segments. The value is represented as an 8-bit unsigned integer. The minimum value is 1.

The code for this option is 37, and its length is 1.

Code	Len	TTL
37	1	n

7.2. TCP Keepalive Interval Option

This option specifies the interval (in seconds) that the client TCP should wait before sending a keepalive message on a TCP connection. The time is specified as a 32-bit unsigned integer. A value of zero indicates that the client should not generate keepalive messages on connections unless specifically requested by an application.

The code for this option is 38, and its length is 4.

Code	Len	Time			
38	4	t1	t2	t3	t4

7.3. TCP Keepalive Garbage Option

This option specifies the whether or not the client should send TCP keepalive messages with a octet of garbage for compatibility with older implementations. A value of 0 indicates that a garbage octet should not be sent. A value of 1 indicates that a garbage octet should be sent.

The code for this option is 39, and its length is 1.

Code	Len	Value
39	1	0/1

8. Application and Service Parameters

This section details some miscellaneous options used to configure miscellaneous applications and services.

8.1. Network Information Service Domain Option

This option specifies the name of the client's NIS [17] domain. The domain is formatted as a character string consisting of characters from the NVT ASCII character set.

The code for this option is 40. Its minimum length is 1.

Code	Len	NIS Domain Name				
40	n	n1	n2	n3	n4	...

8.2. Network Information Servers Option

This option specifies a list of IP addresses indicating NIS servers available to the client. Servers SHOULD be listed in order of preference.

The code for this option is 41. Its minimum length is 4, and the length MUST be a multiple of 4.

Code	Len	Address 1				Address 2			
41	n	a1	a2	a3	a4	a1	a2	...	

8.3. Network Time Protocol Servers Option

This option specifies a list of IP addresses indicating NTP [18] servers available to the client. Servers SHOULD be listed in order of preference.

The code for this option is 42. Its minimum length is 4, and the length MUST be a multiple of 4.

Code	Len	Address 1				Address 2			
42	n	a1	a2	a3	a4	a1	a2	...	

8.4. Vendor Specific Information

This option is used by clients and servers to exchange vendor-specific information. The information is an opaque object of n octets, presumably interpreted by vendor-specific code on the clients and servers. The definition of this information is vendor specific. The vendor is indicated in the class-identifier option. Servers not equipped to interpret the vendor-specific information sent by a client MUST ignore it (although it may be reported). Clients which do not receive desired vendor-specific information SHOULD make an attempt to operate without it, although they may do so (and announce they are doing so) in a degraded mode.

If a vendor potentially encodes more than one item of information in this option, then the vendor SHOULD encode the option using "Encapsulated vendor-specific options" as described below:

The Encapsulated vendor-specific options field SHOULD be encoded as a sequence of code/length/value fields of identical syntax to the DHCP options field with the following exceptions:

- 1) There SHOULD NOT be a "magic cookie" field in the encapsulated vendor-specific extensions field.
- 2) Codes other than 0 or 255 MAY be redefined by the vendor within the encapsulated vendor-specific extensions field, but SHOULD conform to the tag-length-value syntax defined in section 2.
- 3) Code 255 (END), if present, signifies the end of the encapsulated vendor extensions, not the end of the vendor extensions field. If no code 255 is present, then the end of the enclosing vendor-specific information field is taken as the end of the encapsulated vendor-specific extensions field.

The code for this option is 43 and its minimum length is 1.

Code Len Vendor-specific information

```

+-----+-----+-----+-----+
| 43 |  n |  i1 |  i2 | ...
+-----+-----+-----+-----+
```

When encapsulated vendor-specific extensions are used, the information bytes 1-n have the following format:

Code	Len	Data item	Code	Len	Data item	Code
T1	n	d1 d2 ...	T2	n	D1 D2 ...	...

8.5. NetBIOS over TCP/IP Name Server Option

The NetBIOS name server (NBNS) option specifies a list of RFC 1001/1002 [19] [20] NBNS name servers listed in order of preference.

The code for this option is 44. The minimum length of the option is 4 octets, and the length must always be a multiple of 4.

Code	Len	Address 1				Address 2			
44	n	a1	a2	a3	a4	b1	b2	b3	b4 ...

8.6. NetBIOS over TCP/IP Datagram Distribution Server Option

The NetBIOS datagram distribution server (NBDD) option specifies a list of RFC 1001/1002 NBDD servers listed in order of preference. The code for this option is 45. The minimum length of the option is 4 octets, and the length must always be a multiple of 4.

Code	Len	Address 1				Address 2			
45	n	a1	a2	a3	a4	b1	b2	b3	b4 ...

8.7. NetBIOS over TCP/IP Node Type Option

The NetBIOS node type option allows NetBIOS over TCP/IP clients which are configurable to be configured as described in RFC 1001/1002. The value is specified as a single octet which identifies the client type as follows:

Value	Node Type
0x1	B-node
0x2	P-node
0x4	M-node
0x8	H-node

In the above chart, the notation '0x' indicates a number in base-16 (hexadecimal).

The code for this option is 46. The length of this option is always 1.

Code	Len	Node Type
46	1	see above

8.8. NetBIOS over TCP/IP Scope Option

The NetBIOS scope option specifies the NetBIOS over TCP/IP scope parameter for the client as specified in RFC 1001/1002. See [19], [20], and [8] for character-set restrictions.

The code for this option is 47. The minimum length of this option is 1.

Code	Len	NetBIOS Scope				
47	n	s1	s2	s3	s4	...

8.9. X Window System Font Server Option

This option specifies a list of X Window System [21] Font servers available to the client. Servers SHOULD be listed in order of preference.

The code for this option is 48. The minimum length of this option is 4 octets, and the length MUST be a multiple of 4.

Code	Len	Address 1				Address 2		
48	n	a1	a2	a3	a4	a1	a2	...

8.10. X Window System Display Manager Option

This option specifies a list of IP addresses of systems that are running the X Window System Display Manager and are available to the client.

Addresses SHOULD be listed in order of preference.

The code for the this option is 49. The minimum length of this option is 4, and the length MUST be a multiple of 4.

Code	Len	Address 1				Address 2		
49	n	a1	a2	a3	a4	a1	a2	...

9. DHCP Extensions

This section details the options that are specific to DHCP.

9.1. Requested IP Address

This option is used in a client request (DHCPDISCOVER) to allow the client to request that a particular IP address be assigned.

The code for this option is 50, and its length is 4.

Code	Len	Address			
50	4	a1	a2	a3	a4

9.2. IP Address Lease Time

This option is used in a client request (DHCPDISCOVER or DHCPREQUEST) to allow the client to request a lease time for the IP address. In a server reply (DHCPOFFER), a DHCP server uses this option to specify the lease time it is willing to offer.

The time is in units of seconds, and is specified as a 32-bit unsigned integer.

The code for this option is 51, and its length is 4.

Code	Len	Lease Time			
51	4	t1	t2	t3	t4

9.3. Option Overload

This option is used to indicate that the DHCP "sname" or "file" fields are being overloaded by using them to carry DHCP options. A DHCP server inserts this option if the returned parameters will exceed the usual space allotted for options.

If this option is present, the client interprets the specified additional fields after it concludes interpretation of the standard option fields.

The code for this option is 52, and its length is 1. Legal values for this option are:

Value	Meaning
1	the "file" field is used to hold options
2	the "sname" field is used to hold options
3	both fields are used to hold options

Code	Len	Value
52	1	1/2/3

9.4. DHCP Message Type

This option is used to convey the type of the DHCP message. The code for this option is 53, and its length is 1. Legal values for this option are:

Value	Message Type
1	DHCPDISCOVER
2	DHCPOFFER
3	DHCPREQUEST
4	DHCPDECLINE
5	DHCPACK
6	DHCPNAK
7	DHCPRELEASE

Code	Len	Type
53	1	1-7

9.5. Server Identifier

This option is used in DHCPOFFER and DHCPREQUEST messages, and may optionally be included in the DHCPACK and DHCPNAK messages. DHCP servers include this option in the DHCPOFFER in order to allow the client to distinguish between lease offers. DHCP clients indicate which of several lease offers is being accepted by including this option in a DHCPREQUEST message.

The identifier is the IP address of the selected server.

The code for this option is 54, and its length is 4.

Code	Len	Address			
54	4	a1	a2	a3	a4

9.6. Parameter Request List

This option is used by a DHCP client to request values for specified configuration parameters. The list of requested parameters is specified as *n* octets, where each octet is a valid DHCP option code as defined in this document.

The client MAY list the options in order of preference. The DHCP server is not required to return the options in the requested order, but MUST try to insert the requested options in the order requested by the client.

The code for this option is 55. Its minimum length is 1.

Code	Len	Option Codes		
55	<i>n</i>	c1	c2	...

9.7. Message

This option is used by a DHCP server to provide an error message to a DHCP client in a DHCPNAK message in the event of a failure. A client may use this option in a DHCPDECLINE message to indicate the why the client declined the offered parameters. The message consists of *n* octets of NVT ASCII text, which the client may display on an available output device.

The code for this option is 56 and its minimum length is 1.

Code	Len	Text		
56	<i>n</i>	c1	c2	...

9.8. Maximum DHCP Message Size

This option specifies the maximum length DHCP message that it is willing to accept. The length is specified as an unsigned 16-bit integer. A client may use the maximum DHCP message size option in DHCPDISCOVER or DHCPREQUEST messages, but should not use the option in DHCPDECLINE messages.

The code for this option is 57, and its length is 2. The minimum legal value is 576 octets.

Code	Len	Length	
57	2	11	12

9.9. Renewal (T1) Time Value

This option specifies the time interval from address assignment until the client transitions to the RENEWING state.

The value is in units of seconds, and is specified as a 32-bit unsigned integer.

The code for this option is 58, and its length is 4.

Code	Len	T1 Interval			
58	4	t1	t2	t3	t4

9.10. Rebinding (T2) Time Value

This option specifies the time interval from address assignment until the client transitions to the REBINDING state.

The value is in units of seconds, and is specified as a 32-bit unsigned integer.

The code for this option is 59, and its length is 4.

Code	Len	T2 Interval			
59	4	t1	t2	t3	t4

9.11. Class-identifier

This option is used by DHCP clients to optionally identify the type and configuration of a DHCP client. The information is a string of *n* octets, interpreted by servers. Vendors and sites may choose to define specific class identifiers to convey particular configuration or other identification information about a client. For example, the identifier may encode the client's hardware configuration. Servers not equipped to interpret the class-specific information sent by a client **MUST** ignore it (although it may be reported).

The code for this option is 60, and its minimum length is 1.

Code	Len	Class-Identifier
60	n	i1 i2 ...

9.12. Client-identifier

This option is used by DHCP clients to specify their unique identifier. DHCP servers use this value to index their database of address bindings. This value is expected to be unique for all clients in an administrative domain.

Identifiers consist of a type-value pair, similar to the

It is expected that this field will typically contain a hardware type and hardware address, but this is not required. Current legal values for hardware types are defined in [22].

The code for this option is 61, and its minimum length is 2.

Code	Len	Type	Client-Identifier
61	n	t1	i1 i2 ...

10. Extensions

Additional generic data fields may be registered by contacting:

Internet Assigned Numbers Authority (IANA)
 USC/Information Sciences Institute
 4676 Admiralty Way
 Marina del Rey, California 90292-6695
 or by email as: iana@isi.edu

Implementation specific use of undefined generic types (those in the range 61-127) may conflict with other implementations, and registration is required.

11. Acknowledgements

The authors would like to thank Philip Almquist for his feedback on this document. The comments of the DHCP Working Group are also gratefully acknowledged. In particular, Mike Carney and Jon Dreyer from SunSelect suggested the current format of the Vendor-specific Information option.

RFC 1497 is based on earlier work by Philip Prindeville, with help from Drew Perkins, Bill Croft, and Steve Deering.

12. References

- [1] Droms, R., "Dynamic Host Configuration Protocol", RFC 1531, Bucknell University, October 1993.
- [2] Reynolds, J., "BOOTP Vendor Information Extensions", RFC 1497, USC/Information Sciences Institute, August 1993.
- [3] Croft, W., and J. Gilmore, "Bootstrap Protocol", RFC 951, Stanford University and Sun Microsystems, September 1985.
- [4] Braden, R., Editor, "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, USC/Information Sciences Institute, October 1989.
- [5] Mogul, J., and J. Postel, "Internet Standard Subnetting Procedure", STD 5, RFC 950, USC/Information Sciences Institute, August 1985.
- [6] Postel, J., and K. Harrenstien, "Time Protocol", STD 26, RFC 868, USC/Information Sciences Institute, SRI, May 1983.
- [7] Postel, J., "Name Server", IEN 116, USC/Information Sciences Institute, August 1979.
- [8] Mockapetris, P., "Domain Names - Implementation and Specification", STD 13, RFC 1035, USC/Information Sciences Institute, November 1987.
- [9] Postel, J., "Quote of the Day Protocol", STD 23, RFC 865, USC/Information Sciences Institute, May 1983.

- [10] McLaughlin, L., "Line Printer Daemon Protocol", RFC 1179, The Wollongong Group, August 1990.
- [11] Accetta, M., "Resource Location Protocol", RFC 887, CMU, December 1983.
- [12] Mogul, J. and S. Deering, "Path MTU Discovery", RFC 1191, DECWRL, Stanford University, November 1990.
- [13] Deering, S., "ICMP Router Discovery Messages", RFC 1256, Xerox PARC, September 1991.
- [14] Leffler, S. and M. Karels, "Trailer Encapsulations", RFC 893, U. C. Berkeley, April 1984.
- [15] Hornig, C., "Standard for the Transmission of IP Datagrams over Ethernet Networks", RFC 894, Symbolics, April 1984.
- [16] Postel, J. and J. Reynolds, "Standard for the Transmission of IP Datagrams Over IEEE 802 Networks", RFC 1042, USC/Information Sciences Institute, February 1988.
- [17] Sun Microsystems, "System and Network Administration", March 1990.
- [18] Mills, D., "Internet Time Synchronization: The Network Time Protocol", RFC 1305, UDEL, March 1992.
- [19] NetBIOS Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP transport: Concepts and Methods", STD 19, RFC 1001, March 1987.
- [20] NetBIOS Working Group, "Protocol Standard for a NetBIOS Service on a TCP/UDP transport: Detailed Specifications", STD 19, RFC 1002, March 1987.
- [21] Scheifler, R., "FYI On the X Window System", FYI 6, RFC 1198, MIT Laboratory for Computer Science, January 1991.
- [22] Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC 1340, USC/Information Sciences Institute, July 1992.

13. Security Considerations

Security issues are not discussed in this memo.

14. Authors' Addresses

Steve Alexander
Lachman Technology, Inc.
1901 North Naper Boulevard
Naperville, IL 60563-8895

Phone: (708) 505-9555 x256
EMail: stevea@lachman.com

Ralph Droms
Computer Science Department
323 Dana Engineering
Bucknell University
Lewisburg, PA 17837

Phone: (717) 524-1145
EMail: droms@bucknell.edu