1

Physical Media

The language of the RFC was warm and welcoming.
—KATIE HAFNER AND MATTHEW LYON, *Where Wizards Stay Up Late*

While many have debated the origins of the Internet, it's clear that in many ways it was built to withstand nuclear attack. The Net was designed as a solution to the vulnerability of the military's centralized system of command and control during the late 1950s and beyond. For, the argument goes, if there are no central command centers, then there can be no central targets and overall damage is reduced.

If one can consider nuclear attack as the most highly energetic, dominating, and centralized force that one knows—an archetype of the modern era—then the Net is at once the solution to and inversion of this massive material threat, for it is precisely noncentralized, nondominating, and nonhostile.

The term *protocol* is most known today in its military context, as a method of correct behavior under a given chain of command. On the Internet, the meaning of protocol is slightly different. In fact, the reason why the Internet would withstand nuclear attack is precisely because its internal protocols are the enemy of bureaucracy, of rigid hierarchy, and of centralization. As I show in this chapter, the material substrate of network protocols is highly flexible, distributed, and resistive of hierarchy.

The packet-switching technologies behind the Internet provided a very different "solution" to nuclear attack than did common military protocol during the Cold War. For example, in 1958 the Royal Canadian Air Force and the U.S. Air Force entered into agreement under the North American Aerospace Defense Command (NORAD). NORAD is a radar surveillance system ringing North America that provides early warnings of missile or other air attacks against Canada and the United States. "The command monitors any potential aerospace threat to the two nations, provides warning and assessment of that threat for the two governments, and responds defensively to any aircraft or cruise missile threatening North American airspace."[1] The NORAD system is a centralized, hierarchical network. It contains regional control sectors, all of which are ultimately controlled by the USSPACECOM Command Center at Cheyenne Mountain in Colorado Springs, Colorado. It functions like a wall, not like a meshwork. Faced with a nuclear attack,

---

Epigraph: Katie Hafner and Matthew Lyon, *Where Wizards Stay Up Late: The Origins of the Internet* (New York: Touchstone, 1996), p. 144.

1. *NORAD: Into the 21st Century,* U.S. Government Printing Office (1997-574-974).

NORAD meets force with force. Once the outer protection zone of the land-mass is compromised, the NORAD command is able to scramble defensive air power through a rigidly defined system of command and control that is directed outward from a single source (USSPACECOM), to subservient end-point installations that help resist attack. The specific location of each radar installation is crucial, as is the path of the chain of command. During the Cold War, NORAD was the lynchpin of nuclear defense in North America. It is a "solution" to the nuclear threat.

The Internet system could not be more different. It follows a contrary organizational design. The Internet is based not on directionality nor on toughness, but on flexibility and adaptability. Normal military protocol serves to hierarchize, to prioritize, while the newer network protocols of the Internet serve to *distribute.*

In this chapter I describe exactly what distribution means, and how protocol works in this new terrain of the distributed network.[2] I attempt to show that protocol is not by nature horizontal or vertical, but that protocol is an algorithm, *a proscription for structure* whose form of appearance may be any number of different diagrams or shapes.

The simplest network diagram is the centralized network (see figure 1.1). Centralized networks are hierarchical. They operate with a single authoritative hub. Each radial node, or branch of the hierarchy, is subordinate to the central hub. All activity travels from center to periphery. No peripheral node is connected to any other node. Centralized networks may have more than one branch extending out from the center, but at each level of the hierarchy power is wielded by the top over the bottom.

---

2. The division of network designs into centralized, decentralized, and distributed appears in Paul Baran's *On Distributed Communications: 1. Introduction to Distributed Communications Networks* (Santa Monica, CA: RAND, 1964), p. 2. Baran's diagrams have been copied by many authors since then.

Following William Evan, John Arquilla and David Ronfeldt suggest a topology even simpler than the centralized network. This is the chain or line network: for example, "in a smuggling chain where people, goods, or information move along a line of separate contacts, and where end-to-end communication must travel through the intermediate nodes." See Arquilla and Ronfeldt, *Networks and Netwars: The Future of Terror, Crime, and Militancy* (Santa Monica, CA: RAND, 2001), p. 7.

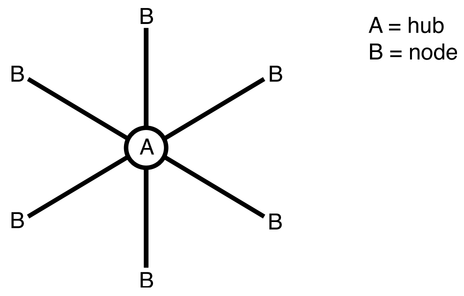


Figure 1.1
A centralized network

The American judicial system, for example, is a centralized network. While there are many levels to the court system, each with its own jurisdiction, each decision of each court can always be escalated (through the appeals process) to a higher level in the hierarchy. Ultimately, however, the Supreme Court has final say over all matters of law.

The panopticon, described in Foucault's *Discipline and Punish,* is also a centralized network. In the panopticon, repurposed by Foucault from the writings of Jeremy Bentham, a guard is situated at the center of many radial cells. Each cell contains a prisoner. This special relationship between guard and prisoner "links the centre and periphery." In it, "power is exercised without division, according to a continuous hierarchical figure" occupying the central hub.[3]

A *de*centralized network is a multiplication of the centralized network (see figure 1.2). In a decentralized network, instead of one hub there are many hubs, each with its own array of dependent nodes. While several hubs exist, each with its own domain, no single zenith point exercises control over all others.

There are many decentralized networks in the world today—in fact, decentralized networks *are the most common diagram of the modern era.*

One example is the airline system. In it, one must always travel through certain centralized hub cities—generally in the Midwest or central areas of the United States. Direct nonstop service is only possible if one happens to

3. Michel Foucault, *Discipline and Punish,* trans. Alan Sheridan (New York: Vintage, 1997), p. 197.

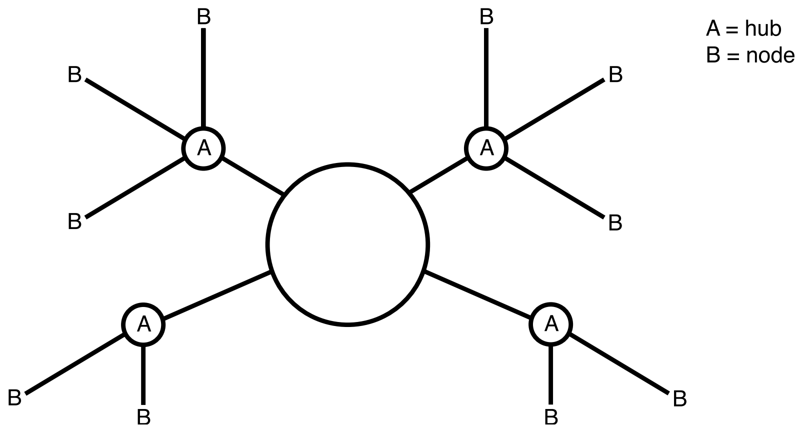


Figure 1.2
A decentralized network

be traveling from one hub to another (or if one pays a premium for special routes).

For the airline system, the decentralized network is the solution to multiplicity, albeit a compromise between the needs of the passenger and the needs of the airlines. There are far too many airports in the country to allow for nonstop service between each and every city; however, it would be inefficient to route every passenger through a single, Midwestern hub (e.g., consider a flight from North Carolina to Maine).

The third network diagram, the one that interests me most here, is called the distributed network.[4] The emergence of distributed networks is part of a larger shift in social life. The shift includes a movement away from central

4. In *Networks and Netwars,* Arquilla and Ronfeldt call this third network topology an "all-channel" network "where everybody is connected to everybody else" (p. 8). However their all-channel network is not identical to a distributed network, as their senatorial example betrays: "an all-channel council or directorate" (p. 8). Truly distributed networks cannot, in fact, support all-channel communication (a combinatorial utopia), but instead propagate through outages and uptimes alike, through miles of dark fiber (Lovink) and data oases, through hyperskilled capital and unskilled laity. Thus distribution is similar to but not synonymous with all-channel, the latter being a mathematical fantasy of the former.

bureaucracies and vertical hierarchies toward a broad network of autonomous social actors.

As Branden Hookway writes: "The shift is occurring across the spectrum of information technologies as we move from models of the global application of intelligence, with their universality and frictionless dispersal, to one of local applications, where intelligence is site-specific and fluid."[5] Computer scientists reference this historical shift when they describe the change from linear programming to *object-oriented* programming, the latter a less centralized and more modular way of writing code. This shift toward distribution has also been documented in such diverse texts as those of sociologist Manuel Castells, American Deleuzian Hakim Bey, and the Italian "autonomist" political movement of the 1970s. Even harsh critics of this shift, such as Nick Dyer-Witheford, surely admit that the shift is taking place. It is part of a larger process of postmodernization that is happening the world over.

What is the nature of these distributed networks? First, distributed networks have no central hubs and no radial nodes. Instead each entity in the distributed network is an autonomous agent.

A perfect example of a distributed network is the rhizome described in Deleuze and Guattari's *A Thousand Plateaus.* Reacting specifically to what they see as the totalitarianism inherent in centralized and even decentralized networks, Deleuze and Guattari instead describe the rhizome, a horizontal meshwork derived from botany. The rhizome links many autonomous nodes together in a manner that is neither linear nor hierarchical. Rhizomes are heterogeneous and connective, that is to say, "any point of a rhizome can be connected to anything other."[6] They are also multiple and asymmetrical: "[a] rhizome may be broken, shattered at a given spot, but it will start up again on one of its old lines, or on new lines."[7] Further, the rhizome has complete disregard for depth models, or procedures of derivation. As Deleuze and Guattari write, a rhizome "is a stranger to any idea of genetic axis

5. Branden Hookway, *Pandemonium: The Rise of Predatory Locales in the Postwar World* (New York: Princeton Architectural Press, 1999), pp. 23–24.

6. Gilles Deleuze and Félix Guattari, *A Thousand Plateaus,* trans. Brain Massumi (Minneapolis: University of Minnesota Press, 1987), p. 7.

7. Deleuze and Guattari, *A Thousand Plateaus,* p. 9.

or deep structure."[8] Trees and roots, and indeed "[a]ll of arborescent culture"[9] is rejected by the rhizome. Summarizing the unique characteristics of the rhizome—and with it the distributed network—Deleuze and Guattari write:

- [U]nlike trees or their roots, the rhizome connects any point to any other point . . .
- The rhizome is reducible neither to the One nor the multiple. . . . It is composed not of units but of dimensions, or rather directions in motion.
- It has neither beginning nor end, but always a middle (*milieu*) from which it grows and which it overspills.
- Unlike a structure, which is defined by a set of points and positions, with binary relations between the points and biunivocal relationships between the positions, the rhizome is made only of lines . . .
- Unlike the tree, the rhizome is not the object of reproduction . . .
- The rhizome is an antigenealogy. It is short-term memory, or antimemory.
- The rhizome operates by variation, expansion, conquest, capture, offshoots.
- The rhizome is an acentered, nonhierarchical, nonsignifying system without a General and without an organizing memory or central automation.[10]

If diagrammed, a distributed network might look like figure 1.3. In a distributed network, each node *may* connect to any other node (although there is no requirement that it does). During a node-to-node connection, no intermediary hubs are required—none, not even a centralized switch as is the case in the telephone network. Point "X" may contact "Y" directly via one of several path combinations.

A distributed network is always caught, to use an expression from Deleuze and Guattari, au milieu, meaning that it is never complete, or integral to itself. The lines of a distributed network continue off the diagram. Any subsegment of a distributed network is as large and as small as its parent network. Distribution propagates through rhythm, not rebirth.

8. Deleuze and Guattari, *A Thousand Plateaus,* p. 12.

9. Deleuze and Guattari, *A Thousand Plateaus,* p. 15.

10. Deleuze and Guattari, *A Thousand Plateaus,* p. 21, bulleted format mine.

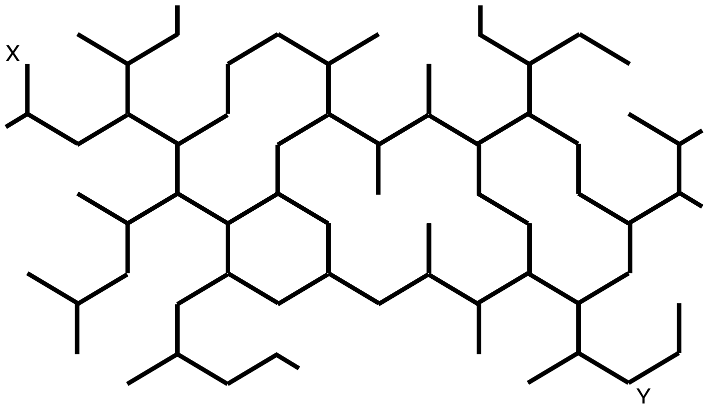
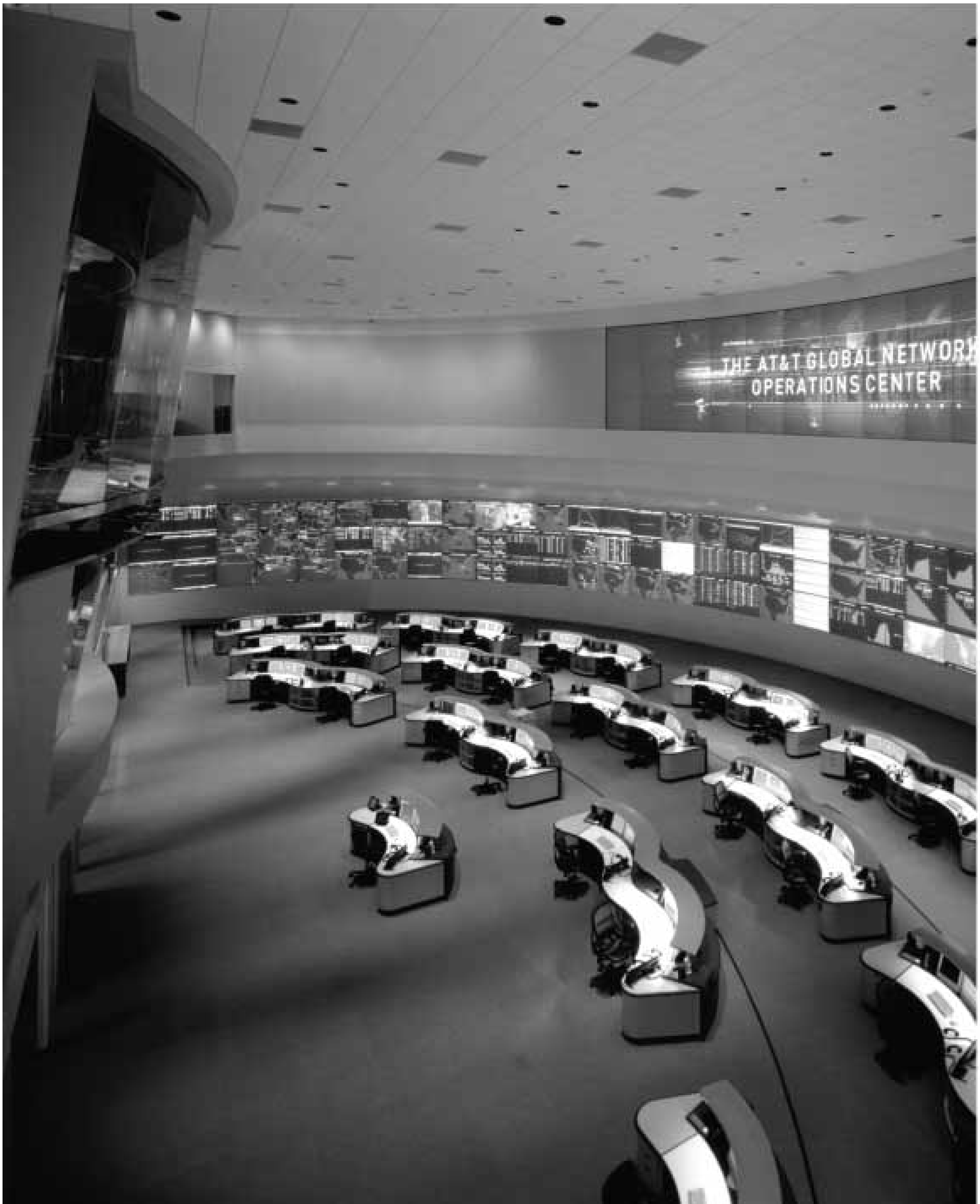


Figure 1.3
A distributed network

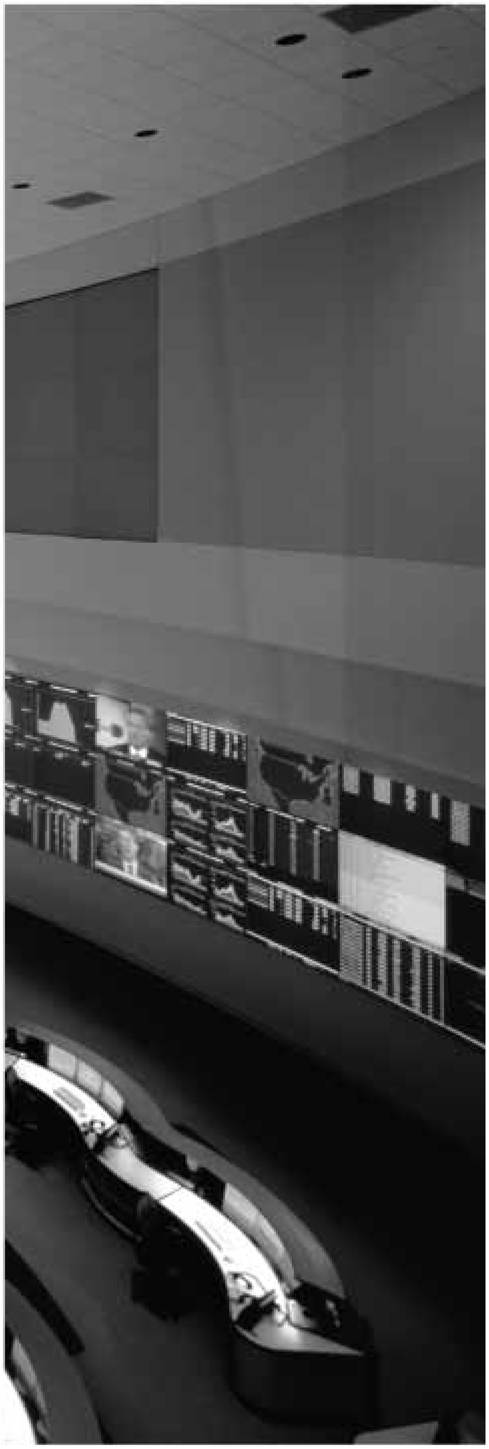
One actually existing distributed network is the Dwight D. Eisenhower System of Interstate & Defense Highways, better known as the interstate highway system. The highway system was first approved by Congress immediately following World War II, but was not officially begun until June 29, 1956, when President Eisenhower signed it into law. (This is exactly the same period during which Internet pioneer Paul Baran began experimenting with distributed, packet-switching computer technologies at the Rand Corporation.[11]) The highway system is a distributed network because it lacks any centralized hubs and offers direct linkages from city to city through a variety of highway combinations.

For example, someone traveling from Los Angeles to Denver may begin by traveling on Interstate 5 north toward San Francisco turning northwest on Interstate 80, or head out on Interstate 15 toward Las Vegas, or even Interstate 40 toward Albuquerque. The routes are varied, not predetermined. If one route is blocked, another will do just as well. These are the advantages of a distributed network.

11. As Hafner and Lyon write: "Baran was working on the problem of how to build communication structures whose surviving components could continue to function as a cohesive entity after other pieces were destroyed." See Katie Hafner and Matthew Lyon, *Where Wizards Stay Up Late,* p. 56.



AT&T Global Network Operation Center (architect: HOK; photo: Peter Paige)



Centrali ation   Decentrali ation

A centralized network consists of a single central power point (a host), from which are attached radial nodes. The central point is connected to all of the satellite nodes which are themselves connected only to the central host. A decentralized network, on the other hand, has *multiple* central hosts, each with its own set of satellite nodes. A satellite node may have connectivity with one or more hosts, but not with other nodes. Communication generally travels unidirectionally within both centralized and decentralized networks: from the central trunks to the radial leaves.

Of course the Internet is another popular and actually existing distributed network. Both the Internet and the U.S. interstate highway system were developed in roughly the same time period (from the late 1950s to the late 1970s), for roughly the same reason (to facilitate mobility and communication in case of war). Later, they both matured into highly useful tools for civilians.

What was once protocol's primary liability in its former military context—the autonomous agent who does not listen to the chain of command—is now its primary constituent in the civil context. The diagram for protocol has shifted from the centralized to the decentralized network, and now finally to the distributed network. Distributed networks have no chain of command, only autonomous agents who operated according to certain pre-agreed "scientific" rules of the system.

For the Internet, these scientific rules are written down. Called protocols, they are available in documents known as RFCs, or "Requests for Comments." Each RFC acts as a blueprint for a specific protocol. It instructs potential software designers and other computer scientists how to correctly implement each protocol in the real world. Far more than mere technical documentation, however, the RFCs are a discursive treasure trove for the critical theorist.

The RFC on "Requirements for Internet Hosts," an introductory document, defines the Internet as a series of interconnected networks, that is, a *network of networks,* that are interconnected via numerous interfacing computers called gateways: "An Internet communication system consists of interconnected packet networks supporting communication among host computers using the Internet protocols . . . The networks are interconnected using packet-switching computers called 'gateways.'"[12] Populating these many different networks are hosts, single computers that are able to send and receive information over the network. According to this RFC, "A host computer, or simply 'host,' is the ultimate consumer of communication services. A host generally executes application programs on behalf of user(s), employing network and/or Internet communication services in support of this function. . . . Internet hosts span a wide range of size, speed, and function.

12. Robert Braden, "Requirements for Internet Hosts," RFC 1122, October 1989, p. 6.

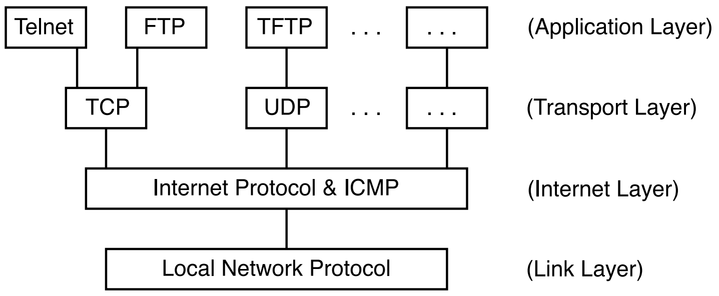


Figure 1.4
Protocol layers

They range in size from small microprocessors through workstations to mainframes and supercomputers."[13] Or, as the RFC on Transmission Control Protocol simply defines it, hosts are "computers attached to a network."[14] If the host is a receiver of information, it is called a client. If it is a sender of information, it is called a server.

In order for hosts to communicate via the Internet, they must implement an entire suite of different protocols. Protocols are the common languages that all computers on the network speak. These component protocols act like layers. Each layer has a different function (see figure 1.4). Considered as a whole, the layers allow communication to happen.

The RFC on "Requirements for Internet Hosts" defines four basic layers for the Internet suite of protocols: (1) the application layer (e.g., telnet, the Web), (2) the transport layer (e.g., TCP), (3) the Internet layer (e.g., IP), and (4) the link (or media-access) layer (e.g., Ethernet).

These layers are nested, meaning that the application layer is encapsulated within the transport layer, which is encapsulated with the Internet layer, and so on.

This diagram, minus its "layer" captions, appears in RFC 791. The four layers are part of a larger, seven-layer model called the OSI (Open Systems Interconnection) Reference Model developed by the International Organization for Standardization (ISO). Tim Berners-Lee, inventor of the Web, uses a

13. Braden, "Requirements for Internet Hosts," pp. 6–7.

14. Jonathan Postel, "Transmission Control Protocol," RFC 793, September 1981, p. 7.

slightly different four-layer model consisting of "the transmission medium, the computer hardware, the software, and the content." Yochai Benkler, from whom Lawrence Lessig has drawn, uses instead a three-layer model consisting of a physical layer, a code layer, and a content layer. Lev Manovich uses an even simpler, two-layer model consisting of a "cultural" layer comprised of "the encyclopedia and the short story; story and plot; composition and point of view; mimesis and catharsis; comedy and tragedy," and a "computer" layer comprised of computer languages, variables, functions, packets, and other code elements.[15]

Consider an average telephone conversation as an analogy. There are several protocols at play during a telephone call. Some are technical, some social. For example, the act of listening for a dial tone and dialing the desired phone number can be considered to be in a different "layer" than the conversation itself.

Furthermore, the perfunctory statements that open and close a telephone conversation—"Hello," "Hi, this is . . . ," "Well, I'll talk to you later," "Okay, goodbye," "Bye!"—are themselves not part of the normal conversation "layer" but are merely necessary to establish the beginning and end of the conversation.

The Internet works the same way. The application layer is like the conversation layer of the telephone call. It is responsible for the content of the specific technology in question, be it checking one's email, or accessing a Web page. The application layer is a *semantic* layer, meaning that it is responsible for preserving the content of data within the network transaction. The application layer has no concern for larger problems such as establishing net-

15. For these references, see Jonathan Postel, "Internet Protocol," RFC 791, September 1981, p. 5; Tim Berners-Lee, *Weaving the Web* (New York: HarperCollins, 1999), pp. 129–130; Yochai Benkler's "From Consumers to Users: Shifting the Deeper Structures of Regulation Toward Sustainable Commons and User Access," *Federal Communications Law Journal* 52 (2000), pp. 561–579; and Lev Manovich, *The Language of New Media* (Cambridge: MIT Press, 2001), p. 46. The critical distinction is that the OSI model, my preferred heuristic, considers everything to be code and makes no allowances for special anthropomorphic uses of data. This makes it much easier to think about protocol. The other models privilege human-legible forms, whose reducibility to protocol is flimsy at best.

work connections, or actually sending data between those connections. It simply wants its "conversation" to work correctly.

The transport layer is one step higher in the hierarchy than the application layer. It has no concern for the content of information (one's email, one's Web page). Instead, the transport layer is responsible for making sure that the data traveling across the network arrives at its destination correctly. It is a social layer, meaning that it sits halfway between the content or meaning of the data being transferred and the raw act of transferring that data. If data is lost in transit, it is the transport layer's responsibility to resend the lost data.

Thus, in our hypothetical telephone conversation, if one hears static on the line, one might interject the comment, "Hello . . . Are you still there?" This comment is *not* part of the conversation layer (unless your conversation happens to be about "still being there"); rather, it is an interstitial comment meant to confirm that the conversation is traveling correctly across the telephone line. The opener and closer comments are also part of the transport layer. They confirm that the call has been established and that it is ready for the conversation layer, and conversely that the conversation is finished and the call will be completed.

The third layer is the Internet layer. This layer is larger still than both the application and transport layers. The Internet layer is concerned with one thing: the actual movement of data from one place to another. It has no interest in the content of that data (the application layer's responsibility) or whether parts of the data are lost in transit (the transport layer's responsibility).

The fourth layer, the link layer, is less important to my study. It is the hardware-specific layer that must ultimately encapsulate any data transfer. Link layers are highly variable due to the many differences in hardware and other physical media. For example, a telephone conversation can travel just as easily over normal telephone wire as it can over fiber-optic cable. However, in each case the technology in question is radically different. These technology-specific protocols are the concern of the link (or media-access) layer.

The different responsibilities of the different protocol layers allow the Internet to work effectively. For example, the division of labor between the transport layer and the Internet layer, whereby error correction is the sole responsibility of the transport layer and routing (the process by which data is "routed," or sent toward its final destination) is the sole responsibility of the Internet layer, creates the conditions of existence for the distributed network.

Thus, if a router goes down in Chicago while a message is en route from New York to Seattle, the lost data can be resent via Louisville instead (or Toronto, or Kansas City, or Lansing, or myriad other nodes). It matters not if the alternate node is smaller or larger, or is on a different subnetwork, or is in another country, or uses a different operating system.

The RFCs state this quality of flexibility very clearly:

> A basic objective of the Internet design is to tolerate a wide range of network characteristics—e.g., bandwidth, delay, packet loss, packet reordering, and maximum packet size. Another objective is robustness against failure of individual networks, gateways, and hosts, using whatever bandwidth is still available. Finally, the goal is full "open system interconnection": an Internet host must be able to interoperate robustly and effectively with any other Internet host, across diverse Internet paths.[16]

As long as the hosts on the network conform to the general suite of Internet protocols—like a lingua franca for computers—then the transport and Internet layers, working in concert, will take care of everything.

The ultimate goal of the Internet protocols is totality. The virtues of the Internet are robustness, contingency, interoperability, flexibility, heterogeneity, pantheism. Accept everything, no matter what source, sender, or destination.

TCP is the most common protocol in the transport layer. It works very closely with the IP to ensure that the data sent via IP arrives correctly. TCP creates a "virtual circuit" between sender and recipient and uses that imaginary circuit to regulate the flow of information. Where IP is blind to the ultimate integrity of the data it transports (more on IP later), TCP constantly checks to see if the message arrives in one piece. As the RFC specifies, "TCP is used by those applications needing reliable, connection-oriented transport service, e.g., mail (SMTP), file transfer (FTP), and virtual terminal service (Telnet)."[17]

TCP is responsible for the "handshake" that happens between two computers at the moment a connection is established.

---

16. Braden, "Requirements for Internet Hosts," p. 8.

17. Braden, "Requirements for Internet Hosts," p. 82.

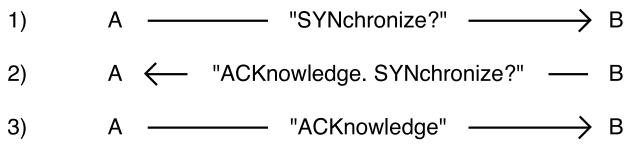


Figure 1.5
Three-way handshake

TCP creates an imaginary circuit between sender and receiver. It "saves state"; that is, it remembers the state of the conversation from moment to moment (something that IP does not do by itself, nor does the other common transport protocol called UDP). This is what the RFC refers to when it describes TCP as "a connection-oriented, end-to-end reliable protocol,"[18] as an example of ongoing "inter-process communication," or as the creation of a "logical circuit" between two computers. The circuit doesn't in fact exist in the real world, but it is created temporarily to connect sender and receiver, in much the same way that a circuit is temporarily created between caller and recipient during a normal telephone conversation (except that with the phone system, the circuit is created by an actual switch, rather than through a distributed connection).

The TCP circuit is created through a three-step process known as a handshake. First, the sender sends a message called a "SYN" (synchronize). Second, the recipient replies with a message called an "ACK" (acknowledge) and initiates its own SYN request. Finally, the original sender acknowledges the recipient's SYN by sending its own ACK (see figure 1.5). After this three-way handshake is complete—(1) "Hello!" (2) "Hi. How are you?" (3) "I'm fine thanks"—the connection is established and normal communication may begin.

The primary value of TCP is its robust quality. TCP allows communication on the Web to be very reliable: Information is monitored during transport and is re-sent if lost or corrupted.

As a system this robustness is achieved by following a general principle: "Be conservative in what you do, be liberal in what you accept from others."[19]

18. Postel, "Transmission Control Protocol," p. 1.

19. Postel, "Transmission Control Protocol," p. 14.

This means that TCP hosts should liberally accept as much information as possible from other, foreign devices. But if any of the information is corrupted, the "conservative" host will delete the information and request a fresh copy be re-sent. As the RFC notes, the goal of TCP is "robustness in the presence of communication unreliability and availability in the presence of congestion."[20]

TCP's partner protocol is IP. TCP and IP work together to create a protocol suite, referred to simply as TCP/IP. IP is responsible for one thing: moving small packets of data called "datagrams" from one place to another. As the RFC specifications for IP note, "the internet protocol provides for transmitting blocks of data called datagrams from sources to destinations."[21]

However, in IP there are "no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols"[22] such as TCP. This means that IP simply seals up its datagrams and shoots them out into the ether. It does not wait for any SYNs or ACKs, and it receives no certification that the datagrams have been received (since these are all the responsibilities of the transport layer, TCP). The IP knows that, eventually, its datagrams will arrive at their locations, and if they don't, the transport layer will provide all error correction and send requests for the missing datagrams to be re-sent.

IP is like the engine powering a car—the engine moves the car, but it has no faculties for knowing when and where to steer, or knowing when and where to stop or speed up (these are the responsibilities of the driver). The engine cannot recognize the different between a green and red traffic light. It has no business dealing with things that are outside its protocological purview.

Technically, then, IP is responsible for two things: routing and fragmentation. Routing is the process by which paths are selected for moving data across a network. Since networks are heterogeneous and ever-changing, the route between point A and point B is never fixed but must be rethought each time material wishes to pass over it.

20. Postel, "Transmission Control Protocol," p. 1.

21. Postel, "Internet Protocol," p. 1.

22. Postel, "Internet Protocol," p. 1.

This flexible routing system is achieved through a "hopping" process whereby data is passed from computer to computer in sequence. None of the computers in the chain of hops knows definitively where the desired destination lies. But they do know in which general direction the destination is. They pass their datagrams to the computer that lies in the "general direction" of the destination. Each computer en route maintains a cache containing information about which of its neighbors lie in which general direction. Each node in the network knows not where the final destination is, but simply which direction, or "next-hop," will get it closer to its destination. If the next-hop proves to be faulty, then the intermediary gateway alerts the source computer and the source computer updates its next-hop cache.

Thus, if Chicago is the next-hop for a message leaving New York en route to Seattle, and Chicago goes down, then Louisville becomes New York's next-hop for Seattle. Later, if Chicago is reinstated and becomes the best routing option again, New York updates its cache accordingly.

The next-hop strategy means that no single node on the Internet knows definitively where a destination is, merely that it is "over there." Each node does know the exact location of every node *it is connected to,* and may pass its messages to whatever machine is closest to "over there." After enough hops in the right direction, the destination machine will no longer be "over there" but will actually be the next-hop for the router currently carrying the data, and the data will be delivered. In this way the message hops around until it arrives in the immediate vicinity of its destination, whereby the exact location of the destination is in fact known and final delivery is possible.

Each datagram is given a number called a "time-to-live." This number designates the maximum number of hops that that datagram is able to take before it is deleted. At each hop, the time-to-live is decreased by one. If the time-to-live reaches zero, the routing computer is obligated to delete the datagram. This ensures that datagrams will not hop around the network indefinitely, creating excess congestion.

The second responsibility of the Internet Protocol is fragmentation. When messages are sent across the network, they are inevitably too large to be sent in one piece. Hence, each message is fragmented, or disintegrated into several small packets, before it is sent. Each small packet is sent over the network individually. At the end, the packets are collected and reassembled to recreate the original message. This process is called fragmentation.

Each physical network has its own personalized threshold for the largest packet size it can accommodate. Thus, no single fragmentation recipe will work for every network. Some, like large freeways, will accommodate large packets, while others, like small back roads, will accommodate only small packets.

But if a message starts its journey as large packets, it cannot be stymied mid-journey if it happens to come upon a foreign network that only accommodates small packet sizes. Refragmentation may be necessary en route. Thus, if a message starts off being fragmented into large packets (e.g., if it is traveling over a fiber-optic cable), it may need to refragment itself mid-journey if it encounters a medium-sized pipe (e.g., a telephone line) somewhere en route. IP can deal with this contingency. Fragmentation allows the message to be flexible enough to fit through a wide range of networks with different thresholds for packet size.

Whenever a packet is created via fragmentation, certain precautions must be taken to make sure that it will be reassembled correctly at its destination. To this end, a header is attached to each packet. The header contains certain pieces of vital information such as its source address and destination address. A mathematical algorithm or "checksum" is also computed and amended to the header. If the destination computer determines that the information in the header is corrupted in any way (e.g., if the checksum does not correctly correlate), it is obligated to delete the packet and request that a fresh one be sent.

At this point, let me pause to summarize the distinct protocological characteristics of the TCP/IP suite:

- TCP/IP facilitates peer-to-peer communication, meaning that Internet hosts can communicate directly with each other without their communication being buffered by an intermediary hub.
- TCP/IP is a distributed technology, meaning that its structure resembles a meshwork or rhizome.
- TCP/IP is a universal language, which if spoken by two computers allows for internetworking between those computers.
- The TCP/IP suite is robust and flexible, not rigid or tough.
- The TCP/IP suite is open to a broad, theoretically unlimited variety of computers in many different locations.

- The TCP/IP protocol, and other protocols like it, is a *result* of the action of autonomous agents (computers).

Each of these characteristics alone is enough to distinguish protocol from many previous modes of social and technical organization. Together they compose a new, sophisticated system of distributed control.

Not every protocol is concerned with the process of peer-to-peer communication as are TCP and IP. DNS, or Domain Name System, is a protocol with a very simple, but different, mandate. DNS is responsible for translating Internet addresses from names to numbers.

While many computer users are familiar with the "dot-com" style of writing Internet addresses (e.g., www.superbad.com or www.rhizome.org), computers themselves use a numerical moniker instead, called an IP address. IP addresses are written as a group of four numbers separated by dots (e.g., 206.252.131.211). While it is very difficult for humans to remember and use such numbers, it is very easy for computers. "The basic problem at hand," writes DNS critic Ted Byfield, is "how we map the 'humanized' names of DNS to 'machinic' numbers of the underlying IP address system."[23] Computers understand numbers more easily, humans understand words. Thus, before each and every transaction on the World Wide Web, one's hand-typed Web address must first be translated to an IP address before the computer can do its work:

```
www.rhizome.org ↔ 206.252.131.211
```

This translation is called "resolution" and it is the reason why DNS exists. If DNS had never been developed, Internet addresses would look more like long telephone numbers or postal codes. Instead they look like long words.

Prior to the introduction of DNS in 1984, a single computer, called a *name server,* held all the name-to-number conversions. They were contained in a single text file. There was one column for all the names and another for all the numbers—like a simple reference table. This document, called HOSTS.TXT,

23. Ted Byfield, "DNS: A Short History and a Short Future," *Nettime,* October 13, 1998.

lived in Menlo Park, California, at the Network Information Center of the Stanford Research Institute (SRI-NIC).[24] Other computers on the Internet would consult this document periodically, downloading its information so that their local reference tables would carry the most up-to-date data. The entire system of naming referred to in this file was called the *name space.*

This early system was a centralized network, par excellence, with SRI-NIC at the center. However as the Internet grew larger this single, central node became incompatible with the nature of the network: "The toll on SRI-NIC, in terms of the network traffic and processor load involved in distributing the file, was becoming unbearable. . . . Maintaining consistency of the files across an expanding network became harder and harder. By the time a new HOSTS.TXT could reach the farthest shores of the enlarged ARPAnet, a host across the network had changed addresses, or a new host had sprung up that users wanted to reach."[25]

To solve this problem, computer scientist Paul Mockapetris designed a new system, a decentralized database of name/number mappings called DNS (see figure 1.6). The new system, still in place today, operates like an inverted tree:

> The domain name space is a tree structure. Each node and leaf on the tree corresponds to a resource set (which may be empty). . . . The domain name of a node or leaf is the path from the root of the tree to the node or leaf. By convention, the labels that compose a domain name are read left to right, from the most specific (lowest) to the least specific (highest).[26]

The tree structure allows Mockapetris to divide the total name space database into more manageable and decentralized zones through a process of hierarchization. As Mockapetris writes, "approaches that attempt to collect a consistent copy of the entire database will become more and more expensive

24. See Paul Albitz and Cricket Liu, *DNS and BIND, Third Edition* (Sebastopol, CA: O'Reilly, 1998), p. 3.

25. Albitz and Liu, *DNS and BIND,* pp. 3–4.

26. Paul Mockapetris, "Domain Names—Concepts and Facilities," RFC 882, November 1983, p. 6.

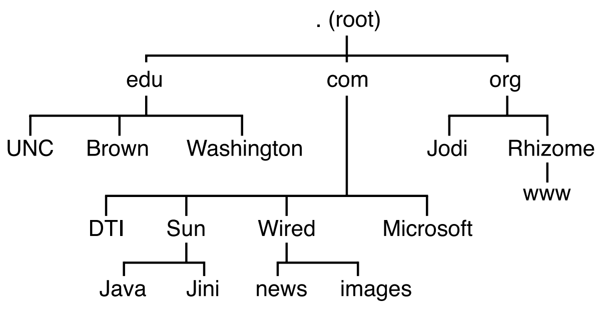


Figure 1.6
Domain Name System (DNS)

and difficult, and hence should be avoided."[27] Instead each portion of the database is delegated outward on the branches of the tree, into each leaf.

At the top of the inverted tree sit the so-called root servers, represented by a single dot (".") They have authority over the *top-level domains* (TLDs) such as "com," "net," "edu," and "org." At each branch of the tree, control over a different zone of the name space is delegated to a server that is lower on the tree. Thus, in order to resolve the address "www.rhizome.org," one must first ask the root server where to find the "org" zone. The root server replies with an authoritative answer about where to find the "org" name server. Then, the "org" name server is queried and replies with the answer for where to find the "rhizome" host within the "org" zone. Finally, the "rhizome" name server is queried, and replies with the numerical address for the "www" computer that lives within the "rhizome" domain.

Like this, the process starts at the most general point, then follows the chain of delegated authority until the end of the line is reached and the numerical address may be obtained. This is the protocol of a decentralized network.

In DNS, each name server can reply only with authoritative information about the zone immediately below it. This is why the system is hierarchical. But each name server can *only* know authoritative information about the zone immediately below it. The second, or third, or even fourth segment down the branch has been delegated to other name servers. This is why the system is decentralized.

27. Mockapetris, "Domain Names—Concepts and Facilities," p. 2.

The more central name servers that are closer to the root of the tree cannot tell you authoritative information about the computers at the ends of the branches, but they *can* tell you who they have delegated such information to and where to find the delegates.

As mentioned in the introduction to this book, protocol is based on a contradiction between two opposing machinic technologies: One radically distributes control into autonomous locales (exemplified here by TCP and IP), and the other focuses control into rigidly defined hierarchies (exemplified here by DNS). There are other important conclusions that one may derive from the preceding discussion of protocol.

First, as the discussion of DNS suggests, protocol is a universalizing system. Ted Byfield writes that what is unique to the DNS is

> its historical position as the first "universal" addressing system—that is, a naming convention called upon . . . to integrate not just geographical references at every scale . . . but also commercial language of every type (company names, trademarks, jingles, acronyms, services, commodities), proper names (groups, individuals), historical references (famous battles, movements, books, songs), hobbies and interests, categories and standards (concepts, specifications, proposals) . . . the list goes on and on.[28]

DNS is the most heroic of human projects; it is the actual construction of a single, exhaustive index for all things. It is the encyclopedia of mankind, a map that has a one-to-one relationship with its territory. Thus, as I demonstrate in chapter 2, DNS is like many other protocols in that, in its mad dash toward universality, it produces sameness or consistency where originally there existed arbitrariness. As the saying goes, apples and oranges are not comparable in the "real world," but in the DNS system they are separated by a few binary digits. DNS is not simply a translation language, *it is language.* It governs meaning by mandating that anything meaningful must register and appear somewhere in its system. This is the nature of protocol.

Second, as the discussion of TCP/IP shows, protocol is materially immanent. That is, protocol does not follow a model of command and control that places the commanding agent outside of that which is being commanded. It

28. Byfield, "DNS."

is endogenous. (This is a departure from the more hierarchical definition of protocol used by the military where control is exercised from without.)

For example, the protocological manipulation of an HTML object by an HTTP object begins first with the parsing of the HTML object:

```
<html>
<body>
Hello World!
</body>
</html>
```

The creation of a special HTTP *header* that derives from the original object is attached to the beginning of it and describes it in various ways:

```
HTTP/1.1 200 OK
Date: Sun, 28 Jan 2001 20:51:58 GMT
Server: Apache/1.3.12 (Unix)
Connection: close
Content-Type: text/html

<html>
<body>
Hello World!
</body>
</html>
```

The header contains various pieces of information about the HTML object such as the date the file was last modified (line 2), the make and model of the server offering the file (line 3), and the type of content it is (in this case, it is text-based HTML [line 5]).

The HTTP object, then, is simply the HTML object plus its new HTTP header, all wrapped up into a new form and separated by a blank line. The new header is prefixed to the original content, becoming part of its material body. But, since the HTTP header is nothing but a description of the material contents of the HTML object, the larger protocol (HTTP) is simply a way of rewriting the smaller one (HTML)—the smaller data object is encapsulated by the larger one. In doing so, the HTML object is immanently transformed—*its actual data is prefixed by another unit of data*—to function within the larger context of HTTP.

Another conclusion is that, while protocol is immanent to a particular set of data, *protocological objects never contain their own protocol.* Thus, TCP/IP houses HTTP, which houses HTML, which houses ASCII text, etc. New headers are added at each level, but in terms of content, protocols are never continuous with themselves.

At each phase shift (i.e., the shift from HTML to HTTP, or from HTTP to TCP), one is able to identify a data object from the intersection of two articulated protocols. In fact, since digital information is nothing but an undifferentiated soup of ones and zeros, data objects *are nothing* but the arbitrary drawing of boundaries that appear at the threshold of two articulated protocols.[29] In order to see HTML, one must actually view it as it intersects with HTTP. Otherwise, one looks at HTML and sees nothing but its own internal protocols: text and markup tags.

A last point, something that should be of particular interest to critical theorists, is that protocol is *against interpretation.* This is to say that protocol does little to transcode the meaning of the semantic units of value that pass in and out of its purview. It encodes and decodes these values, yes, but such transformations are simply trivial mathematics and do not affect meaning in the same way that a Hollywood film may affect the meaning of femininity, or a police officer walking the beat may affect the meaning of power in public space. Protocols do not perform any interpretation themselves; that is, they encapsulate information inside various wrappers, while remaining relatively indifferent to the content of information contained within.

The consequences of this are legion. It means that protocological analysis must focus not on the sciences of meaning (representation/interpretation/reading), but rather on the sciences of possibility (physics or logic), which I address in more detail in chapter 5 on hacking.

The limits of a protocological system and the limits of *possibility* within that system are synonymous.

29. This is similar to Manovich's principle of "modularity" in which every new media object is made up of independent parts, which themselves are unique independent objects. It is, in a sense, objects all the way down. See Lev Manovich, *The Language of New Media,* pp. 30–31.

To follow a protocol means that everything possible within that protocol is already at one's fingertips. Not to follow means no possibility. Thus, protocological analysis must focus on the possible and the impossible (the envelope of possibility), not a demystification of some inner meaning or "rational kernel" within technology. *Protocol is a circuit, not a sentence.*

In this chapter on physical media I have tried to describe protocol from the perspective of its real material substrate. I described the distributed network and positioned protocol as a unique governing principle within that network. I highlighted the TCP/IP suite of Internet protocols and DNS as the two most important theoretical moments for protocol—one protocol radically distributes control into autonomous agents, the other rigidly organizes control into a tree-like decentralized database.

Next, I move beyond the hard science of protocol and begin to consider it from the perspective of form. That is: How does protocol function, not as a material machine, but as an entire formal apparatus? What techniques are used by and through protocol to create various cultural objects? How can one define protocol in its most abstract sense?

These are the fundamental questions contained in chapter 2 on form, to which I now turn.