Chapter 1

The Story Of Bridging

> ‘Challenge-and-response’ is a formula describing the free play of forces that provokes new departures in individual and social life. An effective challenge stimulates men to creative action . . .
>
> — *Arnold Toynbee*

This set of notes is organized around a description of the *history* of bridges. This chapter also describes some of the stimuli that lead to innovation, and introduces some of the people that produced such inventions.

Arnold Toynbee [TC72] describes history using a challenge-response theory in which civilizations either grow or fail in response to a series of challenges. Similarly, the history of bridges can be described as a series of three challenges that are described in the three sections of this chapter: Ethernets under fire (Section 1.1), wire speed Ethernet forwarding (Section 1.2), and scaling to higher speeds (Section 1.3. The responses to these challenges led to what is now known as 802.1 Spanning Tree bridges [IEE97].

1.1 Challenge 1: Ethernet Under Fire

The first challenge arose in the late 1980’s. Ethernet, invented in the 1970’s as a low-cost, high bandwidth interconnect for Personal Computers, was attacked as behaving poorly at large loads and being incapable of spanning large distances. Recall that if two or more nodes on an Ethernet send data at the same time, a collision occurs on the shared wire. All senders then compute a random retranmission time and retry, where the randomization is chosen to minimize the probability of further collisions.

Theoretical analyses (e.g., [Bux80]) claimed that as the utilization of an Ethernet grew, the effective throughput of the Ethernet dropped to zero as the entire bandwidth was wasted on retransmissions. A second charge against Ethernet was its small distance limit of 1.5 Km, much smaller than the limits imposed by say the IBM Token Ring.

While the limited bandwidth charge turned to be false in practice ([BMK88]), it remained a potent marketing bullet for a long time. The second limited distance charge was, and remains, a true limitation of a single Ethernet. In this embattled position around 1980, network marketing people at Digital Equipment Corporation (DEC) pleaded with their technical folks for a technical riposte to these attacks. Could not their bright engineers find a clever way to "extend" a single Ethernet such that it could become a longer Ethernet with a larger effective bandwidth?

First, it was necessary to discard some unworkable alterntives. Physical layer bit repeaters were unworkable because they did not avoid the distance and bandwidth limits of ordinary Ethernets. Extending an Ethernet using a router did, in theory, solve both problems but introduced two other problems. First, in those stone ages, routers were extemely slow and could hardly keep up with the speed of the Ethernet.

Second, there were at least six different routing protocols in use at that time including IBM's SNA, Xerox's SNS, DECNET, Appletalk etc. Hard as it may seem to believe, the Internet protocols were then only a small player in the marketplace. Thus a router would have to be complex beast capable of routing multiple protocols (as Cisco would do a few years later), or one would have to incur the extra cost of placing multiple routers, one for each protocol. Thus the router solution was considered a non-starter.

Routers interconnect links using information in the routing header, while repeaters interconnect links based on physical layer information such as bits. However, in classical network layering there is an intermediate layer called the Data Link layer. For an Ethernet, the Data Link layer is quite simple and contains a 48-bit unique Ethernet Destination address.[1] Why is it not possible, the DEC group argued, to consider a new form of interconnection based only on the Data Link layer? They christened this new beast a Data Link layer relay or a *bridge*.

Let us take an imaginary journey into the mind of Mark Kempf, an engineer in the Advanced Development Group at DEC, who invented bridges around 1980. Undoubtedly, something like the drawing shown in Figure 1.1 was drawn on the chalkboard in Mark's cubicle in Tewksbury, MA. Figure 1.1 shows two Ethernets connected together by a bridge; the lower Ethernet line contains stations A and B, while the upper Ethernet contains station C.

[1]Note that Ethernet 48-bit address have no relation to 32-bit Internet addresses.

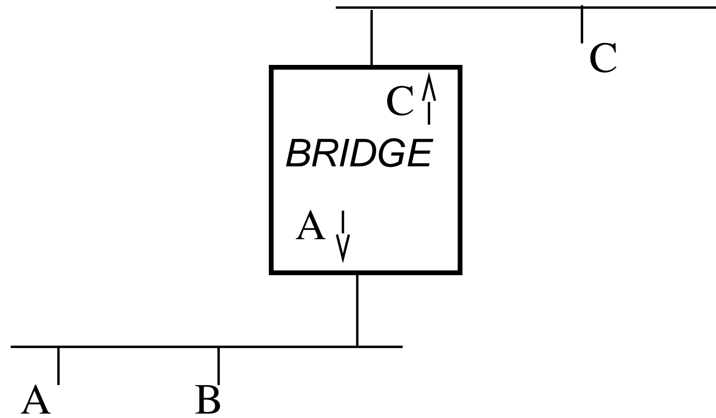

Figure1.1: Towards designing a bridge connecting two Ethernets

The bridge should make the two Ethernets look like one big Ethernet, so that when A sends an Ethernet packet to C it magically gets to C without A having to even know there is a bridge in the middle. Perhaps Mark reasoned as follows in his path to a final solution:

Packet Repeater: Suppose A sends a packet to C (on the lower Ethernet) with destination address C and source address A. Assume the bridge picks up the entire packet, buffers it, and waits for a transmission opportunity to send it on the upper Ethernet. This avoids the physical coupling between the collision resolution processes on the two Ethernets that would be caused by using a bit repeater. Thus the distance span increases to 3 Km, but the effective bandwidth is still that of one Ethernet because every frame is sent on both Ethernets.

Filtering Repeater: The frame repeater idea causes needless waste (**P1**) in Figure 1.1 when A sends a packet to B by sending the packet unnecessarily on the upper Ethernet. This waste can be avoided if the bridge has a table that maps station addresses to Ethernets. For example, in Figure 1.1 suppose the bridge has a table that maps A and B to the lower Ethernet and C to the upper Ethernet. Then on receipt of a packet from A to B on the lower Ethernet, the bridge need not forward the frame because the table indicates that the destination B is on the same Ethernet the packet was received on.

If say a fraction p of traffic on each Ethernet is to destinations on the same Ethernet (locality assumption), then the overall bandwidth of the two Ethernet system becomes $(1 + p)$ times the bandwidth of single Ethernet. This follows because the fraction p can be simultaneously sent on both Ethernets, increasing overall bandwidth by this fraction. Hence *both* bandwidth and distance increase. The only trouble is to figure out how the mapping table is built.

Filtering Repeater with Learning: It is infeasible to have a manager build a mapping

table for a large bridged network. Can the table be built automatically? One aspect of our principle **P13** (exploit degrees of freedom) is Polya's [Pol57] problem solving question: "Have you used all the data?" So far, the bridge has looked only at *destination addresses* to forward the data. Why not also look at *source addresses*? When receiving a frame from A to B, the bridge can look at the source address field to realize that A is on the lower Ethernet. Over time, the bridge will learn the ports through which all active stations can be reached.

Perhaps Mark rushed out after his insight shouting "Eureka". But he still had to work out a few more issues. First, because the table is initially empty, bridges must forward a packet, perhaps unnecessarily, when the location of the destination has not been learnt. Second, to handle station movement table entries must be timed out if the source address is not seen for some time period T. Third, the entire idea generalizes to more than two Ethernets connected together without cycles, to bridges with more than two Ethernet attachments, and to links other than Ethernets that carry destination and source addresses. But there was a far more serious challenge that needed to be resolved.

1.2 Challenge 2: Wire-Speed Forwarding

When the idea was first proposed, some doubting Thomas at DEC noticed a potential flaw. In Figure 1.1 suppose that A sends 1000 packets to B, and then A follows this burst by sending say 10 packets to C. The bridge receives the thousand packets, buffers them, and begins to work on forwarding (actually discarding) them. Suppose the time that the bridge takes to look up its forwarding table is twice as long as the time it takes to receive a packet. Then after a burst of 1000 back-to-back packets arrive, a queue of 500 packets from A to B will remain as a backlog of packets that the bridge has not even examined.

Since the bridge has a finite amount of buffer storage for say 500 packets, when the burst from A to C arrives they may be dropped without examination because the bridge has no more buffer storage. This is ironic because the packets from A to B that are in the buffer will be dropped after examination, but the bridge has dropped packets from A to C that needed to be forwarded. One can change the numbers used in this example but the bottom line is as follows. If the bridge takes more time to forward a packet than the minimum packet arrival time, there are always scenarios in which packets to be forwarded will be dropped because the buffers are filled with packets that will be discarded.

The critics were quick to point out that routers did not have this problem[2] because routers

[2]Oddly enough even routers have the same problem to allow routers to distinguish important packets from less important ones in times of congestion, but this was not taken seriously in the 1980's.

only dealt with packets that were addressed to the router. Thus if a router were used, the router Ethernet interface would not even pick up packets destined to B, avoiding this scenario.

To finesse this issue and avoid interminable arguments, Mark proposed doing an implementation that would do *wire speed forwarding* between two Ethernets. In other words, the bridge would lookup the destination address in the table (for forwarding) and the source address (for learning) in the time it took for a minimum sized packet to arrive on an Ethernet. Given a 64 byte minimum packet, this left 51.2 usec to forward a packet. Since a two port bridge could receive a minimum size packet on each of its Ethernets every 51.2 usec, this actually translated into doing two lookups (destination and source) every 25.6 usec.

It is hard to appreciate today, when wire speed forwarding has become commonplace, how astonishing this goal was in the early 1980's. This is because in those days one would be fortunate to find an interconnect device (e.g., router, gateway) that worked at kilobit rates, let alone at 10 Mbit/sec. Impossible, many thought. To prove them wrong, Mark built a prototype as part of the Advanced Development Group in DEC. A schematic of his prototype, which became the basis for the first bridge, is shown in Figure 1.2.

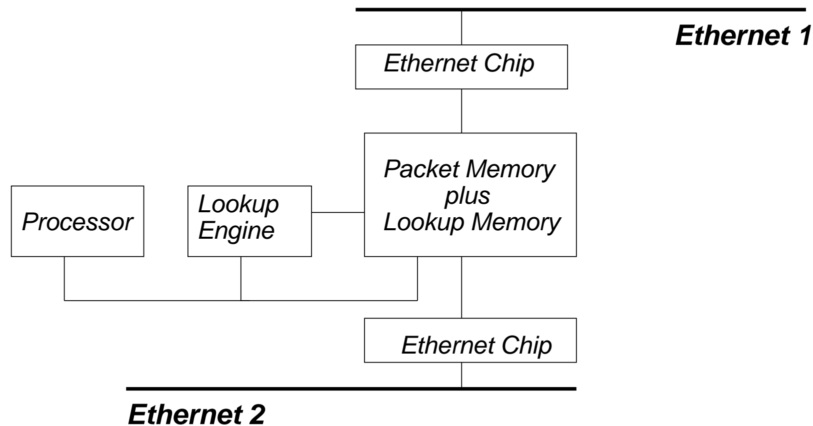

Figure1.2: Implementation of the first Ethernet to Ethernet bridge

The design of Figure 1.2 consists of a processor (the first bridge used a Motorola 68000), two Ethernet chips (the first bridge used AMD Lance chips), a lookup chip (this is described in more detail below), and a 4-ported shared memory. The memory could be read and written by the processor, the Ethernet chips, and the lookup engine.

The data flow through the bridge was as follows. Imagine a packet P sent on Ethernet 1. Both Ethernet chips were set in "promiscuous mode" whereby they received all packets. Thus the bits of P are captured by the upper Ethernet chip and stored in the shared memory

in a receive queue. The processor eventually reads the header of P, extracts the destination address D and gives it to the lookup engine.

The lookup engine looks up D in a database also stored in the shared memory and returns the port (upper or lower Ethernet) in around 1.3 usec. If the destination is on the upper Ethernet, the packet buffer pointer is moved to a free queue, effectively discarding the packet; otherwise, the buffer pointer is moved to the transmit queue of the lower Ethernet chip. The processor also provides the source address S in packet P to the lookup engine for learning.

His design paid careful attention to algorithmics in at least three areas to achieve wire speed forwarding at a surprisingly small manufacturing cost of around 1000 dollars.

- **Architectural Design:** To minimize cost, the memory was cheap DRAM with a cycle time of 100 nsec that was used for packet buffers, scratch memory, and for the lookup database. The 4-port memory (including the separate connection from the lookup engine to the memory) and the busses were carefully designed to maximize parallelism and minimize interference. For example, while the lookup engine worked on doing lookups to memory, the processor continued to do useful work. Note that the processor has to examine the receive queues of both Ethernet chips in dovetailed fashion to check for packets to be forwarded from either the top or bottom Ethernets. Careful attention was paid to memory bandwidth including the use of page mode

- **Data Copying:** The Lance chips used DMA to place packets in the memory without processor control. When a packet was to be forwarded between the two Ethernets, the processor only flips a pointer from the receive queue of one Ethernet chip to the transmit queue of the other processor.

- **Control Overhead:** As with most processors, the interrupt overhead of the 68000 was substantial. To minimize this overhead, the processor used polling, staying in a loop after a packet interrupt, servicing as many packets as arrive in order to reduce context switching overhead . When the receive queues are empty, the processor moves to doing other chores such as processing control and management traffic. The first data packet arrival after such an idle period interrupts the processor but this interrupt overhead is overhead over the entire batch of packets that arrive before another idle period begins.

- **Lookups:** Very likely, Mark went through the eight cautionary questions found. First, to avoid any complaints, he decided to use binary search (**P15**, efficient data structures) for lookup because of its determinism. Second, having a great deal of software experience before he began designing hardware, he wrote some sample 68000 code and determined

that software binary search lookup was the bottleneck (**Q2** in and would exceed his packet processing budget of 25.6 usec.

Eliminating the destination and source lookup would allow him to achieve wire speed forwarding (**Q3**). Recall that each iteration of binary search reads an address from the database in memory, compares it with the address that must be looked up, and uses this comparison to determibe the next address to be read. Using added hardware (**P5**), the comparison can be implemented using combinatorial logic and so a first order approximation of lookup time is the number of DRAM memory accesses.

As the first product aimed for a table size of 8000[3], this required $\log_2 8000$ memory accesses of 100 nsec each, yielding a lookup time of 1.3 usec. Given that the processor does useful work during the lookup, two lookups for source and destination easily fit within a 25.6 usec budget (**Q4**).

To decide whether custom hardware is worthwhile, Mark found that the lookup chip could be cheaply and quickly implemented using a PAL (programmable array logic). His initial prototype met wire speed tests constructed using logic analyzers.

The 68000 software, written by Bob Shelley, also had to be carefully constructed to maximize parallelism. After the prototype was built, Tony Lauck, then head of DECNET, was worried that bridges would not work correctly if they were placed in cyclic topologies. For example, if two bridges are placed between the same pair of Ethernets, messages sent on one Ethernet will be forwarded at wire speed in the loop between bridges. In response, Radia Perlman, then the DEC routing architect, invented her celebrated *spanning tree* algorithm. The algorithm ensures that bridges compute a loop-free topology by having redundant bridges turn off appropriate bridge ports.

While you can read up on the design of the spanning tree algorithm in Radia's book [Per92], it is interesting to note that there was initial resistance to implementing her algorithm which appeared to be "complex" compared to simple, fast bridge data forwarding. However, the spanning tree algorithm used control messages called Hellos that are not processed in real-time.

A simple back-of-the-envelope calculation by Tony Lauck related the number of instructions used to process a hello (at most 1000), the rate of hello generation (specified at that time to be once every second), and the number of instructions per second of the Motorola 68000 (around 1 million). Lauck's vision and analysis carried the day, and the spanning tree algorithm was implemented in the final product.

[3] This allows a bridged Ethernet to only have 8000 stations. While this is probably sufficient for most customer sites, later bridge implementations raised this figure to 16K and even 64K

Manufactured at a cost of one thousand dollars, the first bridge was initially sold at a markup of around eight, ensuring a handsome profit for DEC when sales initially climbed. Mark Kempf was awarded U.S. Patent 4,597,07 titled "Bridge circuit for interconnecting networks" in 1986. DEC made no money from patent licensing, choosing instead to promote the IEEE 802.1 bridge interconnection standards process.

Together with the idea of self-learning bridges, the spanning tree algorithm has also passed into history. Ironically, one of the first customers complained that their bridge did not work correctly; field service later determined that the customer had connected two bridge ports to the same Ethernet, and the spanning tree had (rightly) turned the bridge off! While features like autoconfigurability and provable fault-tolerance have only recently been added to Internet protocols, they were part of the bridge protocols in the 1980's.

The success of Ethernet bridges led to proposals for several other types of bridges connecting other local area networks and even wide-area bridges. The author even remembers working with John Hart (who went on to become CTO of 3Com) and Fred Baker (who went on to become a Cisco Fellow) on building satellite bridges that could link geographically distributed sites. While some of the initital enthusiasm to extend bridges to supplant routers was somewhat extreme, bridges found their most successful niche in cheaply interconnecting similar local area networks at wire speeds.

However, after the initial success of 10 Mbps Ethernet bridges, enginers at DEC began to worry about bridging higher speed LANs. In particular, DEC decided, perhaps unwisely, to concentrate their high speed interconnect strategy around 100 Mbps FDDI token rings [UoNH01]. Thus in the early 1990's, engineers at DEC and other companies began to worry about building a bridge to interconnect two 100 Mpbs FDDI rings. Could wire speed forwarding, and especially exact match lookups, be made ten times faster?

1.3 Challenge 3: Scaling Lookups to Higher Speeds

First, let's understand why binary search forwarding *does not* scale to FDDI speeds. Binary search takes $\log_2 N$ memory accesses to lookup a bridge database, where N is the size of the database. As bridges grew popular, marketing feedback indicated that the database size needed to be increased from 8K to 64K. Thus using binary search, each search would take 16 memory accesses. Doing a search for the source and destination addresses using 100 nsec DRAM would then take 3.2 usec.

Unlike Ethernet where small packets are padded to ensure a minimum size of 64 bytes, a minimum size packet consisting of FDDI, routing and transport protocol headers could be

as small as 40 bytes. Given that a 40 byte packet can be received in 3.2 usec at 100 Mbps, two binary search lookups would use up all of the packet processing budget for a single link, leaving no time for other chores such as inserting and removing from link chip queues.

One simple approach to meet the challenge of wire speed forwarding is to retain binary search but to use faster hardware. In particular faster SRAM could be used to store the database. Given a factor of 5 to 10 decrease in memory access time using SRAM in place of DRAM, binary search will easily scale to wire speed FDDI forwarding.

However, this approach is unsatisfactory for two reasons. First, it is more expensive because SRAM is more expensive than DRAM. Second, using faster memory gets us lookups at FDDI speeds but will not work for the next speed increment (e.g., Gigabit Ethernet). What is needed is a way to reduce the number of memory accesses associated with a lookup so that bridging can scale with link technology.

1.3.1 Scaling via Hashing

In the 1990's, DEC decided to build a fast crossbar switch connecting up to 32 links, called the Gigaswitch [SKz+94]. This chapter concentrates on the bridge lookup algorithms used in the Gigaswitch. The vision of the original designers, Bob Simcoe and Bob Thomas, was to have the Gigaswitch be a switch connecting point-to-point FDDI links without implementing bridge forwarding and learning. Bridge lookups were considered to be too complex at 100 Mbps speeds.

Into the development arena strode a young software designer who changed the product direction. Barry Spinney, who had implemented an Ada compiler in his last job, was determined to do hardware design at DEC. Barry suggested that the Gigaswitch be converted to a bridge interconnecting FDDI Local Area networks. To do so, he proposed designing an FDDI-to-GIGAswitch network controller (FGC) chip on the line cards that would implement a hashing based algorithm for lookups. The Gigaswitch article [SKz+94] states that each bridge lookup makes at most 4 reads from memory.

Now every student of algorithms [CLR90] knows that hashing, on average, is much faster (constant time) than binary search (logarithmic time). However, the same student also knows that hashing is much slower in the worst case, potentially taking linear time because of collisions. How, then, can the Gigaswitch hash lookups claim to take at most 4 reads to memory in the worst case even for bridge databases of size 64K, whereas binary search would require 16 memory accesses?

The Gigaswitch trick has its roots in an algorithmic technique (**P15**) called *perfect hashing* [DKM+88]. The idea is to use a parameterized hash function where the hash function can

be changed by varying some parameters. Then appropriate values of the parameters can be precomputed (**P2a**) to obtain a hash function such that the worst-case number of collisions is small and bounded.

While finding such a good hash function may take (in theory) a large amount of time, this is a good tradeoff because this new station's addresses do not get added to local area networks at a very rapid rate. On the other hand, once the hash function has been picked, lookup can be done at wire speeds.

Specifically, the Gigaswitch hash function treats each 48-bit address as a 47-degree polynomial in the Galois field of order 2, GF(2). While this sounds impressive, this is the same arithmetic used for calculating CRCs; it is identical to ordinary polynomial arithmetic except that all additions are done mod 2. A hashed address is obtained by the equation $A(X) * M(X) mod G(X)$, where G(X) is the irreducible polynomial, $X^{48} + X^{36} + X^{25} + X^{10} + 1$, $M(X)$ is a non-zero, 47-degree programmable hash multiplier, and A(X) is the address expressed as a 47-degree polynomial.

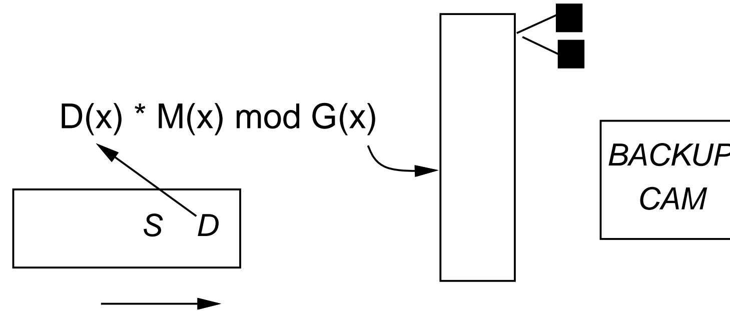

Figure1.3: Gigaswitch hashing uses a hash function with a programmable multiplier, a small balanced binary tree in every hash bucket, and a backup CAM to hold the rare case of entries that result in more than 7 collisions.

The hashed address is 48 bits. The bottom 16 bits of the hashed address is then used as an index into a 64K-entry hash table. Each hash table entry (see Figure 1.3 as applied to the destination address lookup, with $D(x)$ being used in place of $A(x)$) points to the root of a balanced binary tree of height at most 3. The hash function has the property that it suffices to use only the remaining high-order 32 bits of the hashed address to disambiguate collided keys.

Thus the binary tree is sorted by these 32 bit values, instead of the original 48-bit key. This saves 16 bits to be used for associated lookup information. Thus any search is guaranteed to take no more than 4 memory accesses, 1 to lookup the hash table, and 3 more to navigate a height 3 binary tree.

It turns out that picking the multiplier is quite easy in practice. The coefficients of $M(x)$ are picked randomly. Having picked $M(x)$ it sometimes happens that a few buckets have more than 7 colliding addresses. In such a case, these entries are stored in a small hardware lookup database called a CAM (Content Addressable Memory).

The CAM lookup occurs in parallel with the hash lookup. Finally, in the extremely rare case when several dozen addresses are added to the CAM (say when new station addresses are learned that cause collisions), the central processor initiates a rehashing operation and distributes the new hash function to the line cards. It is perhaps ironic that rehashing occured so rarely in practice that one might worry whether the rehashing code was adequately tested!

The Gigaswitch became a successful product,allowing up to 22 FDDI networks to be bridged together together with other link technologies such as ATM. Barry Spinney was assigned U.S. patent 5,920,900 "Hash-based translation method and apparatus with multiple level collision resolution". While techniques based on perfect hashing [DKM+88] have been around for a while in the theoretical comminity, Barry's contribution was to use a pragmatic version of the perfect hashing idea for high speed forwarding.

1.4 Summary

This chapter on exact match lookups is written as a story — the story of bridging. Three morals can be drawn from this story.

First, bridging was a direct response to the challenge of efficiently extending Ethernets without using routers or repeaters; wire-speed forwarding was a direct response to the problem of potentially losing important packets in a flood of less important packets. At the risk of sounding like a self-help book, challenges are best regarded as opportunities and not as annoyances. The mathematician Felix Klein [Bel86] used to say " You must always have a problem: you may not find what you were looking for but you will find something interesting on the way." For example, it is clear that the main reason why bridges were invented — the lack of high performance multiprotocol routers — is *not* the reason why bridges are still useful today.

This brings us to the second moral. Today it is clear that bridges will never displace routers because of their lack of scalability using flat Ethernet addresses, lack of shortest cost routing, etc. However, they remain interesting today because bridges are interconnect devices with better cost-performance and flexibility than routers for interconnecting a small number of similar Local Area Networks. Thus bridges still abound in the marketplace, often refered to as "switches". What many network vendors refer to as a "switch" is a crossbar switch like the

Gigaswitch that is capable of bridging on every interface. A few new features, notably Virtual LANs (VLANs) [Per92], have been added. But the core idea remains the same.

Third, the *techniques* introduced by the first bridge have deeply influenced the next generation of interconnect devices, from core routers to web switches. Recall that Roger Bannister, who first broke the four minute mile barrier, was followed in a few months by several others. In the same way, the first Ethernet bridge was quickly followed by many other wire-speed bridges. Soon the idea began to flow to routers as well. Other important concepts introduced by bridges include the use of memory references as a metric, the notion of trading update time for faster lookups, and the use of minimal hardware speedups. All these ideas carry over into the study of router lookups in the next chapter.

In conclusion, the challenge of building the first bridge stimulated creative actions that went far beyond the first bridge. While wire-speed router designs are fairly commonplace today, it is perhaps surprising that there are products being announced today that claim gigabit wire speed processing rates for such abstruse networking tasks as encryption and even XML transformations.

1.5 Exercises

1. **ARP Caches:** Another example of an exact match lookup is furnished by ARP caches in a router or endnode. In an Internet router, when a packet first arrives to a destination, the router must store the packet and send an ARP request to the Ethernet containing the packet. The ARP request is broadcast to all endnodes on the Ethernet and contains the IP address of the destination. When the destination replies with an ARP reply containing the Ethernet address of the destination, the router stores the mapping in an ARP table, and sends the stored data packet with the destination Ethernet address filled in.

    - What lookup algorithms can be used for ARP caches?
    - Why might the task of storing data packets awaiting data translation result in packet reordering?
    - Some router implementations get around the reordering problem by dropping all data packets that arrive to find that the destination address is not in the ARP table (however, the ARP request is sent out). Explain the pros and cons of such a scheme.

Bibliography

[Bel86] E.T. Bell. *Men of Mathematics.* Touchstone Books, Reissue Edition, 1986.

[BMK88] D.R. Boggs, J.C. Mogul, and C.A. Kent. Measured capacity of an Ethernet: myths and reality. In *Proc. SIGCOMM '88 Symposium on Communications Architectures and Protocols*, volume18, pages 222–34, 1988.

[Bux80] WBux. Local area subnetworks - A performance comparison. In *Proc. Int. Workshop on Local Area Networks, IFIP WG 6.4, Zurich*, 1980.

[CLR90] ThomasH. Cormen, CharlesE. Leiserson, and RonaldL. Rivest. *Introduction to Algorithms.* MIT Press/McGraw-Hill, 1990.

[DKM+88] M.Dietzfelbinger, A.Karlin, K.Mehlhorn, F.auf, dRohnert, and R.Tarjan. Dynamic perfect hashing: Upper and lower bounds. In *29th IEEE Symposium on Foundations of Computer Science*, 1988.

[IEE97] IEEE. Media access control (mac) bridging of ethernet v2.0 in local area networks. In *http://standards.ieee.org/reading/ieee/std/lanman/802.1H-1997.pdf*, i 1997.

[Per92] Radia Perlman. *Interconnections: Bridges and Routers.* Addison Wesley, 1992.

[Pol57] G.Polya. *How to Solve it.* Princeton University Press, 2nd Edition, 1957.

[SKz+94] R.Souza, P.Krishnakumar, C.zveren, R.Simcoe, B.Spinney, R.Thomas, and R.Walsh. Gigaswitch: A high-performance packet switching platform. In *Digital Technical Journal 6(1):9-22, Winter 1994*, 1994.

[TC72] ArnoldJ. Toynbee and Jane Caplan. *A Study of History, abridged version.* Oxford University Press, 1972.

[UoNH01] Inter OperabilityLab Universityof NewHampshire. Fddi tutorials. In *http://www.iol.unh.edu/training/fddi.html*, 2001.