Monitoring Ethernet Connectivity

Thomas L. Rodeheffer
Systems Research Center
HP Laboratories Palo Alto
HPL-2003-160
July 30th , 2003*

Ethernet, network, connectivity, monitoring, probabilistic, testing

Although the Ethernet promises high probability packet delivery between all pairs of hosts, defects in the installation can sometimes prevent some pairs of hosts from communicating very well. This paper describes a method that was developed and used at Carnegie Mellon University during 1984-1985 to monitor the connectivity between all pairs of hosts on the Ethernet. The method is based on sending test packets from a central station through various cyclic routes and relating the results to the likelihood of various defects via a system probability model. The method proved to be quite effective in practice and greatly assisted our support staff in maintaining our Ethernet.

* Internal Accession Date Only
Approved for External Publication

Monitoring Ethernet Connectivity

Thomas L. Rodeheffer

Abstract

Although the Ethernet promises high probability packet delivery between all pairs of hosts, defects in the installation can sometimes prevent some pairs of hosts from communicating very well. This paper describes a method that was developed and used at Carnegie Mellon University during 1984–1985 to monitor the connectivity between all pairs of hosts on the Ethernet. The method is based on sending test packets from a central station through various cyclic routes and relating the results to the likelihood of various defects via a system probability model. The method proved to be quite effective in practice and greatly assisted our support staff in maintaining our Ethernet.

1 Introduction

The Ethernet was invented by Bob Metcalfe and others at the Xerox Palo Alto Research Center [1, 2] and soon developed into an international standard [3]. One aspect that made the Ethernet so attractive was its use of a *passive communications medium*: a thick coaxial cable into which taps could be inserted pretty much at will. And they were. I was a graduate student at Carnegie Mellon University (CMU) in the 1980's as the Ethernet was deployed on campus, mostly using the early Digital-Intel-Xerox (DIX) standard [4]. We observed that as soon as an Ethernet coax came near a group's computers, taps would appear as if by magic. The Ethernet was such an elegantly simple and useful method of connecting computers together that everyone wanted to get on the bandwagon.

The DIX standard contains certain installation requirements designed to avoid signal integrity problems due to electrical reflections [4, section 7.6]. In the early deployment at CMU, these requirements were not always rigorously observed, to say the least. The result was that, sometimes, some pairs of computers could not communicate very well. Now if the defect was persistent and involved computers that often needed to communicate, such as a workstation and its home file server, users would notice and complain and the support staff would know that something was wrong and go fix it. But if the defect was intermittent or involved computers that had had no reason as yet to communicate, no one would know that there was a problem. I recall that one day we discovered that over half of all Ethernet packets sent to the printer were being lost. This problem had been covered up by timeouts and retransmissions at the software connection layer. No one had noticed, except for a vague sense that printing had seemed slow for the past few months.

The Ethernet promises high-probability delivery of packets between all pairs of hosts. Clearly, a method is needed to monitor an Ethernet installation to verify that this promise is being met and, in particular, to identify defects so that they can be fixed. This paper describes a method that was developed and used at CMU during 1984–1985 but never documented.

The rest of the paper is organized as follows. Section 2 describes existing mechanisms for testing Ethernet connectivity. Section 3 presents a system probability model and Section 4 shows how to update the estimated probabilities based on test results. Section 5

gives details about our implementation and Section 6 relates the experience of using our implementation. Section 7 describes ideas for future work. Section 8 gives conclusions.

2 Existing mechanisms

Using a passive communications medium makes for a simple design, but it means that the medium has no ability to monitor its own performance. So we must either employ special equipment (such as scopes) or arrange for the end stations to monitor their performance. Special equipment may be useful for initial debugging, but for monitoring a system's operational behavior, using the system's own equipment is more cost effective. Also, monitoring actual packet delivery between end stations provides an end-to-end check, which is good practice [5].

To verify that stations X and Y can communicate, the straightforward approach is to cause X to send Y a request that causes Y to send X a corresponding response. This is the strategy of Ping [6], which is arguably the network administrator's most often used tool. Ping uses the IP / ICMP ECHO protocol [7], but a similar protocol could easily be designed for the Ethernet link layer. Although theoretically a defect could discriminate between application traffic and ping traffic, causing pings to work even though application traffic does not, in practice such a situation is assumed not to happen to any significant degree. This is a slight compromise with the end-to-end principle, but in practice ping is considered good enough. Hence if we could arrange for every station to ping every other station on a regular basis, any defect in connectivity would be exposed. Figure 1 illustrates this *all-to-all ping* approach.

Unfortunately, the all-to-all ping approach suffers from a serious practical shortcoming, namely, each station must implement a fairly complicated process to ping all other stations. How would this process know the roster of stations to ping? How would it report problems? How would the total network overhead of ping traffic be regulated? These are all aspects of the complexity of managing the operation of a distributed all-to-all ping connectivity test. An Ethernet can encompass hundreds of stations, using dozens of architectures and operating systems, including printers, workstations, routers, servers, sensors, PDAs, and special-purpose or prototype hardware. It is unrealistic to expect that every station would implement a complicated testing process in a conforming and manageable way.

A more practical approach is to require a certain trivial service in each station and employ a centralized process that exploits this service to test network connectivity. A properly-chosen trivial service can be quite effective. For example, Ping is universally useful as an IP network test program because (1) every IP implementation is required to implement the ICMP ECHO protocol, and furthermore, (2) every implementation actually does implement it, because the protocol is so trivial that even the most feeble implementation can handle it. It turns out that the DIX standard includes a trivial service requirement that is precisely what is needed for a centralized connectivity test.

The DIX standard requires that every station implement the Ethernet Configuration Testing Protocol (ECTP) [4, section 8]. Like the ECHO protocol, ECTP requires that the receiving station emit a packet in response to an incoming request packet. Unlike the ECHO protocol, however, an ECTP packet contains a list of stations and the response

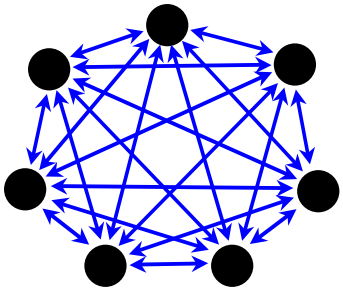


Figure 1: All-to-all ping distributed connectivity test.

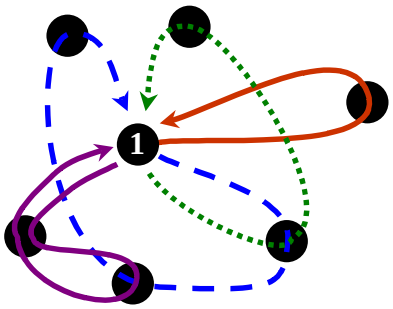


Figure 2: Using ECTP to test various routes from a central station.

goes to the next station on the list, rather than back to the sender. This forwarding action continues station-by-station until the end of the list is reached, whereupon the ECTP packet is at its final destination. By putting itself at the end of the list, the originator of an ECTP packet can route the packet through any sequence of intermediate stations and and end back at the originator. ECTP can be viewed as a generalization of the ECHO protocol. Much the same thing can be accomplished in IP by using an ICMP ECHO REPLY with an explicit source route.

We test a route by sending an ECTP packet along that route. The test is successful if the packet reaches its destination. Figure 2 illustrates some routes that can be tested using ECTP from a central station. For convenience, we number the stations 1, 2, …, *n* with station 1 being the central station. If the network is working properly, that is, packets are being delivered between all pairs of hosts with high probability, then any route that the central station chooses to test will likely succeed. So we can be confident that the network is working properly if the central station successfully tests the 3-hop routes (1, X, Y, 1) for all pairs of stations (X, Y). However, if something is defective, then what the central station observes is that some routes tend not to work. To isolate the broken component, we would like a diagnosis of the form: Y cannot hear X; or, even better: the receiver in Y is broken. The DIX standard outlines a procedure using 2-hop, 3-hop, and 4-hop routes that could be used to obtain such a diagnosis.

The DIX procedure is designed as a tool to diagnose a known or suspected problem, rather than as a process to continually monitor the state of the network. A drawback to using short routes is that what is mostly being tested is the connectivity of the central station itself. For more efficient testing, it would be better if longer routes were used. Long routes test many intermediate stations for every packet sent and received at the central station. If the network is delivering all-pairs connectivity as promised, testing many stations with a long route will likely succeed and with one test the central station will verify that many pairs can communicate.

The question is, what information is obtained when a long route test fails? The following sections present a system model and a method to update the model based on the results of tests.

3 Probabilistic model

When a station is so broken that no packets can get in or out, the problem is obvious and easily noticed. More subtle problems require more sophisticated monitoring. Our experience at CMU showed that common failures were due to defects in a single station or in the interaction between a pair of stations. For example, a station may have a timing or data dependency that causes some packets transmitted or received to be dropped. Or, one station may transmit just poorly enough that some other station cannot always receive its packets. A bad interaction between stations might be due to poor electrical properties of the communications medium resulting from improper installation, but for our present purpose, we assign blame to the pair of stations.

We adopt the simple model that a packet sent from X to Y will be received with probability $P_{X,Y}$. We assume that the probabilities for different pairs are independent, so that the probability that a route test succeeds is simply the product of the probabilities for each hop on the route. It may be objected that such independence is unlikely since, for example, $P_{X,Y}$ and $P_{X,Z}$ both involve the transmitter in X. We could avoid this problem by never choosing routes on which a station appears more than once. In practice, our method chooses such routes infrequently and it was suitably effective even without this refinement.

Figure 3 illustrates the probabilities tabulated as a matrix. Although the model covers the case of a station sending a packet to itself, this aspect is not particularly useful. Most Ethernet stations use software to handle the loop-back case and failure is unlikely.

Assuming that this model captures the essential behavior of the network, diagnosis would be straightforward if we knew the hop transmission probabilities. A station would be suspected of having a broken transmitter if it appeared as the transmitter in many low probabilities (that is, corresponding to a row in the matrix containing many low entries). An isolated low probability might indicate an isolated poor interaction between stations.

Next we describe our method for taking test results and estimating the hop transmission probabilities. This is a non-trivial problem, because the central station cannot perform tests on single hops (except for $P_{1,1}$, which is not useful). Tests must include at least two hops, and three hops are needed to get to probabilities not involving the central station as either transmitter or receiver. As mentioned earlier, for efficient testing, it might be better if even longer routes could be used.

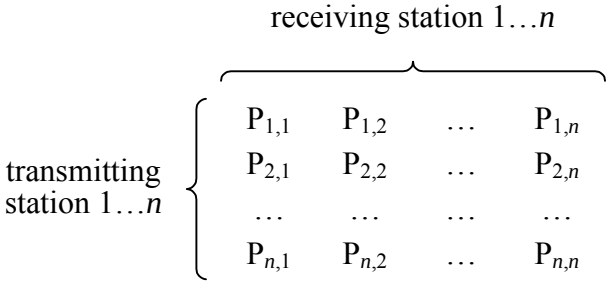


Figure 3: Hop transmission probability matrix P.

4 Estimation method

For the moment, let us assume that the hop transmission probabilities P are stationary. Our method of estimating P proceeds by iteration. We start with an estimate Q, perform a test, construct an updated estimate Q′, and repeat. For each pair (X,Y) we define the estimated hop transmission probability $Q_{X,Y}$ as the ratio of a number of successes $S_{X,Y}$ to a number of tests $T_{X,Y}$:

$$Q_{X,Y} \equiv \frac{S_{X,Y}}{T_{X,Y}}$$

The essential insight involves understanding how to count tests and successes. When we test a route containing several hops, the route will succeed with a probability equal to the product of the hop transmission probabilities. Now if one of the hops has a very low transmission probability, the route will most likely fail and the failure will most likely be due to the bad hop. Testing such a route and getting a failure hardly tells you anything about the good hops. It's not a fair test for the good hops. The good hops only have a chance to show how good they are when the bad hop succeeds.

The way to embody this idea is to count a route test as only a fractional test of each hop on the route. The fraction for a particular hop is precisely the expectation that all the other hops will succeed, that is, the product of the hop transmission probabilities for all the other hops. Then, if the test succeeds, we count one success for each hop on the route. Of course, we do not know the actual hop transmission probabilities, so we compute the expectation based on our current estimate.

Specifically, let us consider a test of a route R. Let Q_R be the estimated probability of success when testing R. Since the probabilities are assumed to be independent, this is just the product of the hop transmission probabilities:

$$Q_R \equiv \prod_{(X,Y)\in R} Q_{X,Y}$$

Then for each hop $(X,Y) \in R$ we compute

$$T'_{X,Y} = T_{X,Y} + \frac{Q_R}{Q_{X,Y}}$$

$$S'_{X,Y} = S_{X,Y} + \begin{cases} 1 & \text{if the test succeeds} \\ 0 & \text{if the test fails} \end{cases}$$

Observe that if the estimates Q are equal to the probabilities P, then the expected value of the update would produce no change in Q.

It would be nice to prove that the estimates computed by this method converge to P or, indeed, converge at all, and how rapidly. This is an open question.

It is clear that the estimates need not converge exactly to P, because every (cyclic) route must use both the transmitter and the receiver of each station involved. Given any station X and a scaling factor α such that

$$\max_{U\neq X} P_{U,X} \le \alpha \le \min_{V\neq X} (P_{X,V})^{-1}$$

we can take P, multiply row X by α and column X by α^{-1}, and get a matrix of probabilities that has the same success probability for any cyclic route as the original matrix P. Figure 4 illustrates this computation, which we call alpha-scaling.

If we consider the graph formed by taking stations as nodes and the logarithm of hop transmission probabilities as edge weights, alpha-scaling is an example of the edge-weight transform

$$w'(u,v) = w(u,v) + h(u) - h(v)$$

where *u* and *v* are nodes and *w*(*u*,*v*) is the edge weight from *u* to *v*. It is well-known that this transform leaves the weights of all cycles unchanged [8, 9].

Although one might try to apply alpha-scaling to produce a "cannonical" or "balanced" matrix of probability estimates, we did not do so. We just took whatever the results were that we got from the estimation method. Further investigation is left for future work.

5 Implementation

We implemented the above probability-based Ethernet connectivity testing method at the Computer Science Department of Carnegie Mellon University during 1984–1985. Our installation was actually a bridged mixture of real and emulated 3Mb/s (experimental) Ethernet, so we designed a 3Mb/s variant of ECTP and managed to get it implemented on all of our mainframes and most of our workstations. A central testing program was written to run on a mainframe and manage the test.

Several features of the implementation should be noted. First, the structure of the estimate update calculations allowed us to employ a particularly simple, "fire and forget" testing process. Second, we had to adjust the implementation to deal with nonsensical and extreme numerical values. Third, we arranged the route selection to get good coverage while limiting the network overhead as well as the burst load on any station.

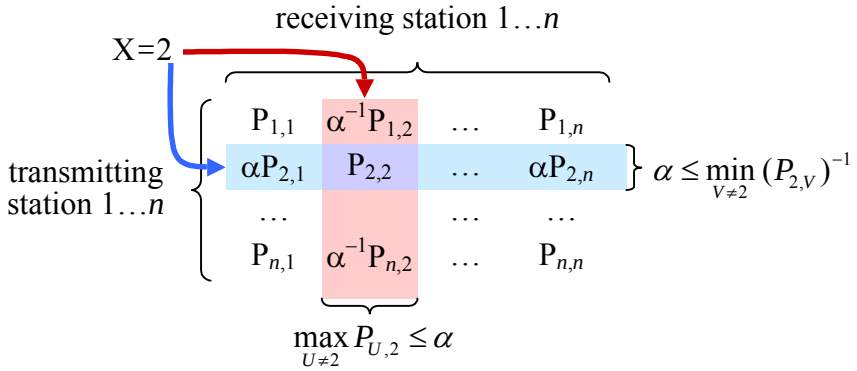


Figure 4: An example of alpha-scaling. The resulting probability matrix has the same success probabilities for any cyclic route as the original matrix P.

5.1 Fire and forget

Although the estimation method was described above as performing one test at a time and updating the probability estimates after each test, there is no need to follow such a rigid schedule. We can count tests when a test packet is launched and count successes some time later, when a test packet returns. So the testing program chooses a route, makes a test packet, launches the packet, updates the hop test counts, and then goes on to choose another route.

Now a test either succeeds or fails. Since a failure adds nothing to the success counts, there is no need to determine if or when a test has failed. On the other hand, if a test succeeds, we get back the test packet, which contains the list of stations on its route. We can just read off the hops and increment the relevant success counts. In either case, there is no need to remember any context from a test. This simplifies the testing program and makes it trivial to have many tests in progress at the same time.

This fire-and-forget strategy has the effect of depressing the probability estimates slightly (essentially assuming that the test will fail) during the interval between launching the test and getting the success. This effect becomes less significant as the total number of tests increases and it does not impair the usefulness of the testing method.

5.2 Numerical fixes

Our testing program was designed to start from scratch and compute hop transmission probabilities over a several hour testing period. We started with an initial estimate of 0.9 transmission probability for each hop by initializing the success and test counters as:

$$S_{X,Y} = 0.9$$
$$T_{X,Y} = 1.0$$

Recall that each test counts as a fractional hop test whereas each success counts as a unit hop success. Since typically many of our hops would be much better than 0.9 probability, early on during the testing process there would often be a lot more successes than expected by the model. We discovered that the ratio of "successes" to "tests" on a hop could sometimes exceed one. Of course, such a value is nonsense as a probability. We fixed this by limiting $Q_{X,Y}$ to a value of at most 1, regardless of $S_{X,Y}$ and $T_{X,Y}$.

We discovered another numerical problem when dealing with hops that were completely broken. The probability estimate for these hops would fall lower and lower and eventually the testing program would suffer an underflow when computing estimated route probabilities. We fixed this by limiting $Q_{X,Y}$ to a value of at least 0.01, regardless of $S_{X,Y}$ and $T_{X,Y}$. This fix has the side-effect that a totally broken hop pulls down all of the other probability estimates slightly in order to keep its own estimate artificially inflated. However, the effect is small and did not impair the usefulness of the testing method in distinguishing between defective and functioning stations.

Incorporating these numerical fixes, the actual computation we used for $Q_{X,Y}$ was

$$Q_{X,Y} \equiv \max\left(\min\left(\frac{S_{X,Y}}{T_{X,Y}}, 1.0 \right), \ 0.01 \right)$$

5.3 Route selection and scheduling

Before the testing program can select any routes to test, it needs a roster of stations. Since we were using 3Mb/s (experimental) Ethernet, which has 8-bit station addresses, testing all possible addresses was one option. This option was especially apt for our installation at CMU, in which over half of all possible addresses were in use. As another option, the testing program was also designed to take a specified list of stations to test. One could also invent various exploration protocols using broadcast ECTP packets; such protocols would be useful on standard Ethernet, which has 48-bit station addresses.

Since the testing program was designed to compute hop transmission probabilities starting from scratch, it seemed a good idea to conduct a number of short route tests in order to get reasonable estimates of hop transmission probabilities before going on to a more extensive period of long route tests. So the testing program started with a phase of all 2-hop tests, then a phase of all 3-hop tests, and then successive phases of longer random tests up to 8 hops.

The 2-hop phase tested the out-and-back routes from the central testing station to each other station. Each route was tested ten times and, to keep from concentrating on any particular station, the various tests were interleaved. The 3-hop phase tested the out-over-and-back routes from the central testing station through each pair of other stations. Again, each route was tested ten times using a schedule to avoid concentrating on any particular station.

The longer tests would select a route by picking stations at random up to the number of hops for the phase. We assumed that random selection would provide enough variance to avoid concentrating on any particular station. If a station never succeeded in any of the 2-hop or 3-hop tests, it would not be included in the longer random tests unless it was on the specified list of stations to test.

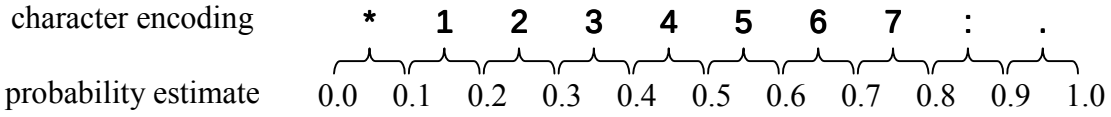
During all phases, a flow-control credit-balance scheme much like a leaky bucket [10] was used to limit network overhead to no more than 100 packets per second. Observe that testing an h-hop route requires h packets to be transmitted on the network if the test succeeds. So when the testing program launched an h-hop route test, it subtracted h credits from the balance. If the result was negative, the program paused for one second and then added 100 credits to the balance. Normal traffic on our network was well over 1000 packets per second, so this limited the overhead to an acceptable percentage.

The total time consumed for all tests was several hours. We assumed that the hop transmission probabilities would be stationary during this interval. Most of the time was spent performing the longer random tests.

6 Experience

During 1984–1985 the CMU Computer Science Department support staff ran the Ethernet testing program several times a month in order to verify that the network was operating properly. If changes in the Ethernet installation were known or suspected, the testing program was run more frequently.

It turned out to be useful to print the probability estimate matrix Q in a condensed tabluar format, encoding each probability estimate as one character as follows:



All the good hops would print as periods, which would fill most of the page. A not-quite-so-good hop would print as a colon, which would draw one's attention unless there were hops that were worse. Anything less than 0.8 was a problem that should be fixed and the printed digit would indicate how serious the problem was. Something less than 0.1 was probably totally busted and printed as an asterisk.

The support staff would take such a printout and look for rows and columns containing excess ink. Unless there were some innocuous explanation, such as a powered-down machine, the relevant hardware would be swapped out or perhaps a tap would be moved, and the next day the Ethernet testing program would be run again to see if the problem was fixed. The support staff loved the Ethernet testing program [11]. Problems were often found and fixed long before any user complained. When the printout showed all dots, the support staff knew that network connectivity was fully operational.

7 Future work

Although development of the Ethernet testing program basically ceased at the point that it became good enough to make the support staff happy, several ideas remain to be explored in the future. First, over long time periods the hop transmission probabilities will not be stationary: ideally, the testing program should be running continually in order to detect such changes. Second, perhaps the testing process could be refined to distinguish between receiver defects and transmitter defects. Third, the effectiveness of testing long routes should be evaluated.

7.1 Non-stationary probabilities and continual testing

Our idea for dealing with non-stationary hop transmission probabilities was to decay the hop success and test counters over time. For example, every minute the testing program could multiply all of its counters by some factor $0 < \beta < 1$, which would cause the counters to decay with a half-life of $\log_\beta(\frac{1}{2})$ minutes. The half-life should be chosen to be long enough to smooth out sampling variation yet long enough to follow significant changes in the probabilities—a half-life of one hour should be about right.

The decay process might interact with the fire-and-forget testing method. If a decay calculation took place while test packets were in flight, the successes would increment as

normal but the number of tests would be decayed. Experimentation would be required to determine if this effect was significant. One solution would be to mark test packets with their "decay epoch" so that the success increments could be adjusted for the decay. Another solution would be simply to stop sending test packets for a few seconds prior to the decay calculation.

Our testing program printed out final results. A continual testing program would have to issue regular reports, and it would also help to highlight significant changes from one report to the next. We never implemented these ideas.

7.2 Distinguishing receiver v.s. transmitter defects

As mentioned in Section 4, since every cyclic route tests both the receiver and transmitter of each involved station, the testing process cannot discriminate between alternate, alpha-scaled versions of the hop transmission probability matrix. Basically, this means that if a station does not work very well, we cannot tell whether the fault lies in its receiver or in its transmitter.

Note that this is not a problem if the station has someone it receives well from and someone it transmits well to. Such a situation bounds α tightly near 1 and does not permit much ambiguity. Distinguishing receiver defects from transmitter defects is only a problem if the station cannot communicate very well with anyone. From a practical perspective, such a station should probably have its hardware swapped out and then the defective hardware examined carefully under laboratory conditions. But we thought it would be nice if we could figure out how to disambigute the defect in situ.

We had two ideas, both of which would involve changing the ECTP protocol and neither of which was very attractive. The first idea was to arrange for a station to tell us how many of our ECTP packets it had received. One could imagine leaving space in the ECTP packet station list for each station on the route to write in its counter value. The counter values would enable the testing program to distinguish receiver defects from transmitter defects. Unfortunately, if other diagnostics were using ECTP while the testing program was running, either the packet counts would be confounded or the ECTP servers would have to be instructed to keep separate counts for each diagnostic. Figuring out how to to keep separate counts would add yet more complexity to the ECTP protocol, so it seemed unattractive.

The other idea was to add an option to the ECTP packet station list whereby a specified station would be required to emit multiple ECTP packets in response to a single received packet. This requirement could take the form of a fork in the station list, whereby the specified station responds with two ECTP packets, each following a potentially different path back to the central station, as illustrated in Figure 5. Alternatively, the requirement could take the form of a duplication, whereby the specified station acts as though it had received two identical ECTP packets, as illustrated in Figure 6. Assuming that all the probabilities are independent, either of these testing options would provide enough statistical information to distinguish receiver defects from transmitter defects. Unfortunately, requiring a station to emit multiple packets in response to a single request is a significant complication to the ECTP protocol and might present difficulties to feeble implementa-

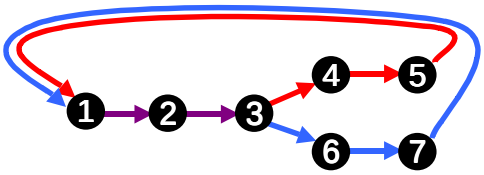


Figure 5: Example test route with a fork at station 3.

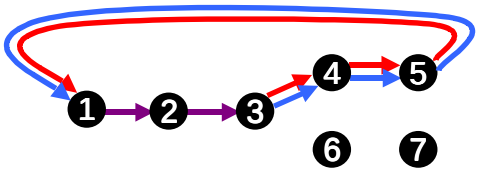


Figure 6: Example test route with a duplication at station 3.

tions. Also, we were skeptical of how independent the probabilities might be for similar or identical packets in close temporal proximity. So this idea also seemed unattractive.

7.3 Effectiveness of testing long routes

One of our motivations for developing the probabilistic method was the claim that testing long routes would be more efficient at monitoring network connectivity than testing short routes. We never really evaluated this claim. Implementing a continual testing process would provide a good platform to perform such an evaluation.

A related evaluation would be to measure the speed of convergence of the probabilistic method. Perhaps an appropriate choice of longer or shorter routes would accellerate convergence. Since the results we were getting were good enough for our support staff, we never put any effort into improving the performance of the testing program.

8 Conclusion

Although the Ethernet promises high probability packet delivery between all pairs of stations, defects in the installation can sometimes prevent some pairs of stations from communicating very well. Such defects must be detected so that they can be fixed before the situation degrades to the point that it impacts users. The simplicity of the Ethernet requires that the stations themselves participate in testing their own connectivity, yet the variety of stations imposes the practical requirement that this participation be extremely simple. Fortunately, the Ethernet Configuration Testing Protocol (ECTP) provides precisely what is needed. ECTP is simple enough for any station to implement and powerful enough to enable a central testing program to test all-pairs connectivity.

Even a well-functioning Ethernet will drop occasional packets, so the testing program must expect some loss. The problem is to distinguish this generalized loss from loss caused by an intermittent defect. The problem is compounded because the testing program cannot directly test sending a packet between two stations; it must instead test an entire route that includes the connection between the two stations as one hop. We developed a probabilistic model and estimation method that enables the testing program to relate the results of route tests to the likelihood of various defects. This method was used at Carnegie Mellon University during 1984–1985. It proved to be quite effective in practice and greatly assisted our support staff in maintaining our Ethernet.

Thanks to Jim Saxe for providing helpful comments on a draft of this paper.

References

1. Robert M. Metcalfe and David R. Boggs. **Ethernet: distributed packet switching for local computer networks**. *Communications of the ACM*, 19(7), July 1976.
2. Robert M. Metcalfe, David R. Boggs, Charles P. Thacker, and Butler W. Lampson. **Multipoint data communication system with collision detection**. U.S. Patent 4,063,220 issued 1977.
3. IEEE Computer Society. **IEEE Std 802.3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications**. Originally published in 1985.
4. Digital Equipment Corporation, Intel Corporation, and Xerox Corporation. **The Ethernet: data link layer and physical layer specifications. Version 2.0**. November 1982.
5. J. H. Saltzer, D. P. Reed, and D. D. Clark. **End-to-end arguments in system design**. *ACM Transactions on Computer Systems*, 2(4), November 1984, pp. 277-288.
6. Mike Muuss. **The story of the PING program**. 1999. http://ftp.arl.mil/~mike/ping.html
7. Internet Engineering Task Force. **RFC 792: Internet control message protocol**. September 1981. http://www.ietf.org/rfc/rfc0792.txt
8. Donald B. Johnson. **Efficient algorithms for shortest paths in sparse networks**. *Journal of the ACM*. 24(1), January 1977, pp. 1-13.
9. Charles E. Leiserson and James B. Saxe. **Retiming synchronous circuitry**. Research Report 13, Digital Systems Research Center, Palo Alto, CA, August 1986. Available from http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-13.html
10. J. S. Turner. **New directions in communications (or which way to the information age?)**. *IEEE Communications Magazine*, 24(10), October 1986, pp. 8-15.
11. Lawrence Butcher. Personal communication, 1985.