A Best-Case Network Performance Model

Steven M. Bellovin*

February 12, 1992

Abstract

Network performance measures usually focus on average throughput. We, however, were concerned with best-case behavior: how fast could a packet traverse the network if there were no contention for resources. By subtracting the path time *to* a node from the path time *through* the node, we were able to develop a simple best-case delay model. This model was sensitive enough to determine the board-level configuration of a router 750 miles away.

1 Introduction

Network performance measures usually focus on average throughput. We, however, were concerned with best-case behavior: how fast could a packet traverse the network if there were no contention for resources. We took advantage of the fact that we had a SPARCstation-1 workstation[1]; those machines have a timer with a resolution of 1μ-second. Our basic strategy was simple. We first measured the elapsed time *to* a node. We then measured the elapsed time to an identical machine, except passing *through* the node in question. The difference in the two times is the transit cost for that node.

Such a strategy sounds simple, and indeed it is. There are, however, a few points to consider. First, of course, one must take measurements of a variety of packet sizes. This has the potential for complication, as host buffer

*AT&T Bell Laboratories. smb@ulysses.att.com

[1]SPARCstation is a trademark of SPARC International, Inc.

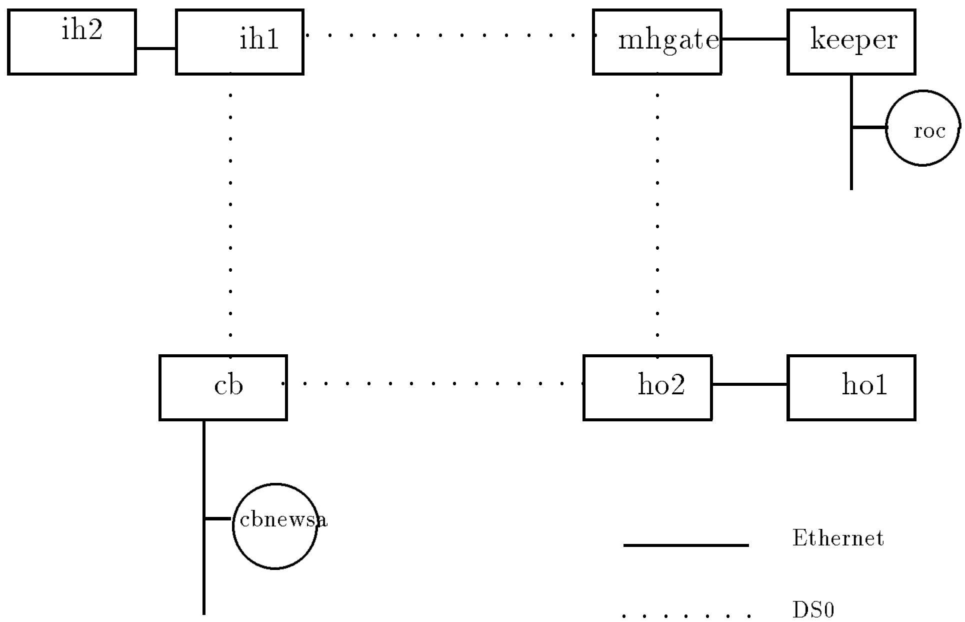


Figure 1: Topology of the network

strategies may differ for packets of different sizes. Indeed, we observed this effect, though it was fairly small.

Measurements such as these must be conducted on identical pairs of nodes. One cannot sensibly compare the times to field the message if the first and second machines are different. Fortunately, the backbone of our internal network is composed of Cisco routers; this provided a good selection of nodes to query. The section we measured is shown in Figure 1. Local area networks are shown as solid lines; long-haul point-to-point links are shown as dotted lines.

We must also consider what values to use for the cost of any operation. Generally, one might pick the mean or median cost. Furthermore, one must discard outliers; network behavior can get amazingly bad at times. Here, however, we are interested in the best possible time, which is a simpler matter. The outliers do not matter; all that we care about is the absolute minimum time. (To be sure, we are making the tacit assumption that in an otherwise-idle network, the best-case time is deterministic. This assumption may not be true in some cases.)

For our measurements, we used the TCP/IP protocol suite. Specifically, we used the the ICMP[Pos81] `echo` and `echo-reply` messages. These are about as low-level as one can get; they are generally processed in the kernel, and hence rely the least on the vagaries of user-level dispatching.

2 Analysis

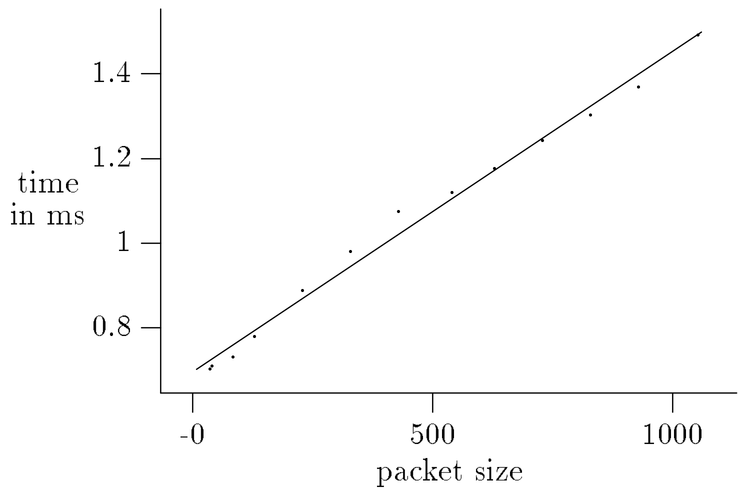
We began by measuring the basic host overhead on `roc`, our workstation, for sending any ICMP packets. We did this by pointing the `ping` command at the local host's own address. There are a number of components comprising the trip time: the user-level time to construct the packet, the basic system call overhead, the time to copy the packet from the user process into a network buffer, IP's overhead to route the packet down to the loopback driver, the trip back up through IP, the time to copy the packet back to user space, and the overhead for decoding the received packet. The return trip includes the same components; for the sake of simplicity, we assume that the two legs are equal, and divide in half. The list is long, but it breaks down into two components: a fixed overhead for sending any ICMP packet, and a per-byte cost for the two copy operations. The time to initialize the packet data areas do not count; the timestamps are computed just before it is sent, and just after it is received.

The observations are shown in Figure 2. We used the S Data Analysis System[BCW88] to compute a least squares fit to the data; this is shown on the graph as a straight line. As can be seen, the fit is fairly good, confirming our analysis. The one-way trip time, in milliseconds, is defined by $.696 + .000757 \times psize$, where $psize$ is the size of the packet.

Note the slight hump from 112 bytes to 512 bytes. This is due to the host buffering strategies. Blocks of up to 112 bytes are copied into `mbuf`s; these are linked together to form the complete packet. However, when the packet size exceeds 512 bytes, a single `cluster` buffer is used. This reduces the overhead for processing the packet.

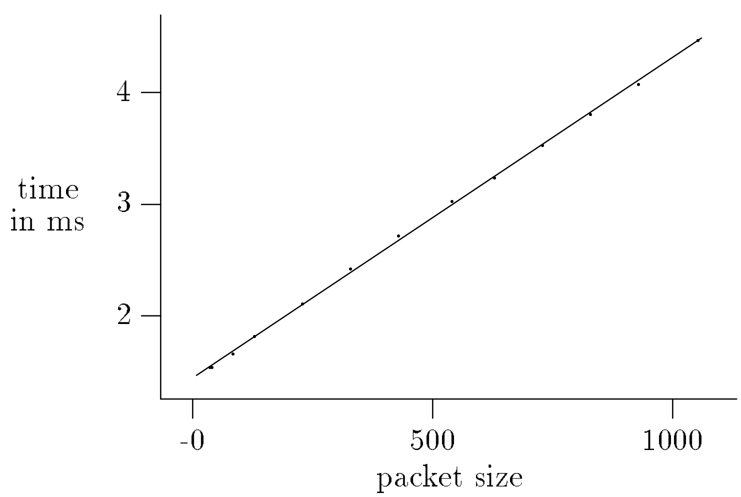
The next step was to measure the response time of the local Cisco router, `keeper`, to `echo` packets. Again, we measured the round-trip time for a variety of packets; see Figure 3. The measured one-way performance was $1.447 + .00287 \times psize$ milliseconds per packet.

When actually sending on physical media, two more factors come into play. The first is the serialization time per byte, i.e., the time to stobe



roc

Figure 2: `Ping` Local loopback timing



keeper

Figure 3: `Ping` times to the first router

the bits onto the wire. Since Ethernet networks[2] run at 10 megabits per second, there is a cost of $.0008 \times psize$ milliseconds per packet. This figure is small but measureable. The second factor is propagation delay. Assuming a maximum cable distance of about 2500 meters, and a signal velocity of $.667c$, the propagation delay is $.0125\mu$-seconds, a number small enough to disregard. We have chosen to disregard the framing overhead for a network type (i.e., for the header, the CRC, etc.), preferring to fold it into the fixed cost of the interface.

We can now work through our basic equation:

$$\texttt{ciscoping} = \texttt{keeper} - (\texttt{roc} + \texttt{ether}).$$

That is, the time required for the router to process the `echo` packet is the observed delay minus the sum of the host overhead and the network delays. Thus,

$$\begin{aligned} \texttt{ciscoping} &= (1.447 + .00287 \times psize - \\ &\quad ((.696 + .000757 \times psize) + .0008 \times psize) \\ &= .751 + .00131 \times psize. \end{aligned}$$

Similar techniques let us calculate the overhead of Cisco's MCI board, a board with up to two Ethernet network ports and two serial interfaces. We measured the times to `ho1` and `ho2` as $6.671 + .15189 \times psize$ and $6.146 + .15105 \times psize$ milliseconds. Calculating

$$\texttt{ciscomci} = \texttt{ho1} - (\texttt{ho2} + \texttt{ether})$$

yields a cost of $.525 + .00004 \times psize$ milliseconds. Note the very low cost per byte, effectively 0. This is in agreement with the architecture of the MCI board; no data copying takes place if off-board forwarding can be avoided.

Note the comparatively high cost to reach `ho2`. At the time of these measurements, the link between it and `mhgate` was a DS0 circuit, i.e., a 56 kilobit per second trunk. Serialization time is .14285 milliseconds per byte, much greater than for Ethernet networks. We should also add in propagation time; however, for this short a distance (about 50 kilometers), it is much less than one millisecond.

[2]Ethernet is a registered trademark of Xerox Corporation.

Table 1: Basic Model Parameters

Tag	Fixed Cost	Cost Per Byte	Component
ethernet	-	.0008	Ethernet cost
ciscoping	.751	.00131	Cisco response time
roc	.696	.000757	local host overhead
att3b2600	2.480	.00263	3B2/600 response time
ciscomci	.525	.00004	Cisco MCI board
ciscobtw	.564	.000582	Cisco — between boards
ciscoold	1.153	.001562	old-style Cisco Ethernet
ciscoser	1.695	.002939	Cisco serial port
ds0	*	.14285	56K bps line

3 Validating the Model

We validated our model by calculating the time to reach `cbnewsa`, an AT&T 3B2/600. A wearisome set of components had to be added together; their values are shown in Table 1. The configurations of the different Cisco routers was determined partly by experimentation, and partly by asking administrators. We have not been able to account for the measured per-byte cost for Cisco serial ports.

The "*" for the fixed cost for `ds0` lines indicates that we must add in speed of light delay between the two points. Note that this is cable distance, approximately 1400 kilometers, not map distance. Furthermore, the lines we use are not "pure" wires. Rather, they are subchannels of a DS1 (1.544 megabit per second) line routed via Chicago, and the multiplexor imposes a delay of about .5 milliseconds.

A summary of the calculation is shown in Table 2. The calculated cost is thus $20.7317 + .3 \times psize$. This is in excellent agreement with the measured cost of $21.872 + 0.296 \times psize$, especially given the uncertainties in cable distance. A graph of the measured versus calculated times is shown in Figure 4.

Table 2: Time to Reach `cbnewsa`

Component	Fixed	Per byte	40 bytes	1024 bytes
9 Ciscos, 3 Ethernets	6.1959	.0110	6.6379	17.5106
Endpoints	3.1758	.00338	3.3110	6.6369
2 DS0 lines		.2858	11.432	292.6352
8 Muxes	4.40	-	4.40	4.40
Propagation	6.96	-	6.96	6.96
One-way Total	20.7317	.3001	12.0072	307.3843

The dashed line is the predicted value.

4 Detecting an Anomaly

An attempt to check our calculations by measuring the time through `ih1` to `ih2` — a pair of nodes about 1025 kilometers from us — failed. We had the following equations:

$$\begin{aligned} \texttt{ih1} &= 13.953 + .1502 \times psize \\ \texttt{ih2} &= 14.517 + .1516 \times psize \\ \texttt{ether} &= .0008 \times psize. \end{aligned}$$

Calculating

$$\texttt{transit} = \texttt{ih2} - (\texttt{ih1} + \texttt{ether})$$

yielded a cost of $.564 + .0006 \times psize$. This seemed wrong; the serial line to `mhgate` was supposedly on the same MCI card as the Ethernet controller, and MCI cards have a per-byte cost an order of magnitude lower. After checking our measurements and calculations, we insisted that the administrator verify his configuration. The result: the node was misconfigured; the two ports were on different boards, thereby necessitating a copy operation.

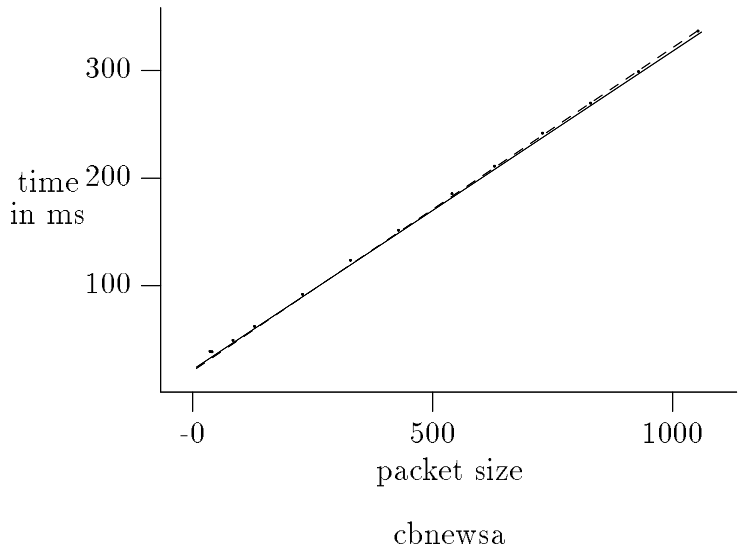


Figure 4: Measured versus calculated times to `cbnewsa`

5 Security Implications

The ability to make accurate predictions and measurements has some benefits for those charged with maintaining the security of a network. Assume that two routers are connected by a DS0 line. In that case, a 1000 byte packet will have a serialization time of 143 milliseconds, a very noticeable figure. An active wiretapper — one who receives each packet, and possibly modifies it before retransmitting — will require a second serialization delay for the retransmission. The alternative — examining each packet on the fly, and sending an HDLC `abort` signal or a bad checksum — will show up in the line's error statistics counter. We conclude that our techniques make it possible to detect active wiretappers.

Why is this useful, since we cannot use these mechanisms to detect passive eavesdroppers? There are a number of cryptographic protocols that can be applied if one can assume that no active wiretapping takes place. For example, the Diffie-Hellman exponential key exchange protocol[DH76] provides excellent security against a listener, but is useless against an opponent who sits in the middle. We can employ this protocol if we can verify that no active wiretaps exist.

6 Conclusions

Our model's accuracy is limited by several factors. One is the minimum system call time, measured at about 45μ-seconds. Given the number of system calls it takes to send and receive a packet, we cannot do better than about 250μ-seconds. There is also sufficient jitter in the dispatching clock that a large number of observations is needed to observe a true minimum value. Even then, we find that different trials can yield values differing by .1 milliseconds or thereabouts.

There is a great deal of uncertainty in estimated wire distances on long-haul links. Apart from phone company routing decisions, we had to contend with the corporate telecommunications strategy, which involved multiplexing DS1 trunks and cross-connecting as needed. Thus, the link from `ho2`, in New Jersey, to `cb`, in Ohio, actually passed near `ih1`, outside Chicago. This is very far from how a crow would fly.

On a positive note, the linear fit seems to be quite good, despite the buffering policies of the host. Our overall accuracy seems to be within a few percent.

Looking at the model analytically, we find that long-haul network delays are dominated by two factors. One is propagation delay; until someone figures out how to increase the speed of light, we can do little beyond avoiding satellite hops. The other is the serialization delay imposed at each router. Since each router must receive the packet in full before forwarding it, the delay is linear in the number of hops. If the packet size is large with respect to the line speed, this becomes a critical factor. Using more packets of a smaller size can be a significant help, because transmission times can then be overlapped. This suggests that the Internet's path MTU discovery algorithm[MD90] may not be an unmixed blessing. Two solutions suggest themselves: increasing the line speed or decreasing the number of hops. The latter may be achievable by using more large-scale communication subnets, after the style of the old ARPANET, rather than dedicated point-to-point links. For networks operating at speeds of a few hundred bits per second or less, the impact may be significant.

References

[BCW88] Richard A. Becker, John M. Chambers, and Allan R. Wilks. *The New S Language.* Wadsworth & Brooks/Cole Advanced Books & Software, 1988.

[DH76] Whitted Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-11:644–654, November 1976.

[MD90] J.C. Mogul and S.E. Deering. *Path MTU Discovery*, November 1990. RFC 1191.

[Pos81] Jon B. Postel. *Internet Control Message Protocol*, September 1981. RFC 792.