Calculating Timeouts for Minimum Cost RPC

Peter B. Danzig

Computer Science Division
University of California, Berkeley
Berkeley, California 94720

Abstract. *We derive expressions for remote procedure call (RPC) retransmission timeout values that minimize the weighted sum of total elapsed time, operating system kernel costs, and network charges. Currently we select RPC retransmission timeouts to minimize network utilization, an operating point chosen with our networks' low bandwidth in mind. When public utilities provided digital service becomes popular, we will routinely make requests to servers across town and across nation and we will pay for each message that we send. These networks may offer various grades of service, each with different loss and delay characteristics. In such an environment, choosing the retransmission timeout and grade of service that minimize our costs is not obvious. Our approach, illustrated with a simple RPC algorithm, can be applied to other remote invocation mechanisms. We can incorporate the expression for the optimal retransmission timeout that our approach generates into the communication's software of distributed operating systems.*

This work has been supported in part by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 4871, monitored by the Naval Electronic Systems Command under Contract No. N00039-84-C-0089 and by an American Electronics Association Faculty Development Fellowship. The views and conclusions contained in this document are those of the author and should not be interpreted as representing official policies, either expressed or implied, of any of the sponsoring organizations or of the U.S. Government.

1. Introduction

Consider a remote procedure call (RPC) between a *requestor* site and a *server* site [1]. Since both the network and the server may lose messages, sometimes the requestor must retransmit its RPC request one or more times. Similarly, since both the network and the requestor may lose messages, sometimes the server must retransmit its reply one or more times. Since the server's service time varies depending on the its load and processor speed, most existing RPC implementations retransmit infrequently, assuming that the network rarely loses messages and that the server will eventually respond. However, in some networks, the probability of message loss can be high. Extended local area networks (LAN) experience buffer overflow at LAN bridges. Internet gateways drop packets due to congestion by caused by stream protocols. Aloha and slotted Aloha networks lack collision detection and lose messages with high probability. The probability of message loss increases with message size, and many request-response protocols can send many kilobytes of data, e.g. encryption key servers, authentication servers, and and network paging servers. Modeling existing networks, we introduce an RPC cost function that accounts

for kernel costs, network costs, and end-to-end message transmission delay, and minimize it as a function of the protocol retransmission timeout.

When public utility provided networks become popular, we can expect to pay for each message we transmit. Perhaps the networks will offer various grades of message reliability and message transmission delay. Unreliable messages cost less than highly reliable messages. Slower messages cost less than faster messages. Modeling these future networks, we minimize our RPC cost function by simultaneously finding the optimal retransmission timeout and grade of service to request.

In Figure 1 we illustrate our RPC retransmission policy. The requestor retransmits its RPC every τ milliseconds until it receives the server's reply. We denote the probability that a message is lost by p, the probability that it is delivered by $q = 1 - p$, the operating system kernel's cost per message by K_r and the network's charge per message by $M_r(y, p)$, where y denotes the message's expected transmission delay. We denote the time between retransmissions, the retransmission timeout, by τ, the server's service time density by $f(t)$, and the expected service time by $\bar{x}$.

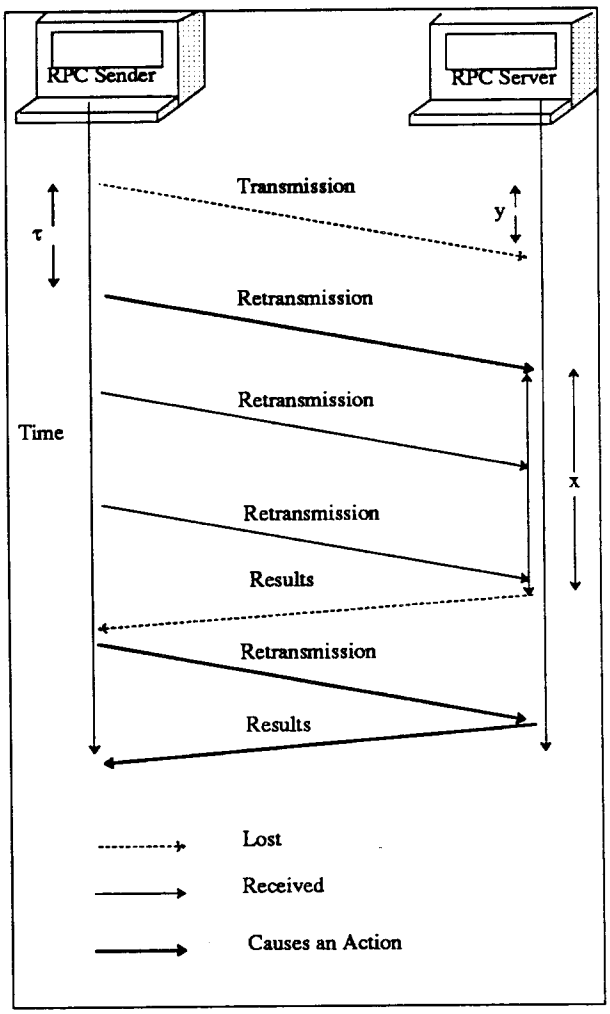

Figure 1. Figure 1. Behavior of the RPC algorithm we consider here.

1.1. RPC cost function

We define the total RPC cost L as the weighted sum of the elapsed time between the requestor's initial transmission and its first reception of the RPC's results, the cost imposed on the operating system kernels by messages sent by both the requestor and the server, and the charge that the network imposes for the transmitted messages. Denote the expected number of message sent by $\bar{n}$, the expected elapsed time by $\bar{t}$, and the expected RPC cost function by L.

$$L = a \cdot \bar{t} + b \cdot \bar{n} K_r + c \cdot \bar{n} \cdot M_r(y, p). \qquad (1)$$

Notice that the weighting constants a, b, and c must translate each component of this sum into some common unit of cost, say milliseconds. Since we can rescale K_r and $M_r(y, p)$, without loss of generality, we set a, b and

c to 1.

1.2. Outline

In Section 2 we calculate optimal timeouts for existing networks in three steps. We first calculate the optimal timeout assuming that message transmission delay is negligible and result messages are not lost. This models busy servers. Busy servers may repeatedly drop requests, but once it receives one, it never drops its response. Next we extend the model to include lost result messages. Finally, we add network transmission delay y. In Section 3 we consider networks with tunable loss and delay parameters, where slow messages cost less than fast messages and less reliable messages cost less than more reliable messages. We simultaneously minimize the RPC cost function L over the retransmission timeout τ and the optimal network transmission time y and loss parameter p to request. In Section 4 we discuss how to incorporate our results in distributed systems. We draw conclusions in Section 5.

2. Optimal retransmission timeout

In this section we derive the optimal retransmission timeout τ_{opt} by evaluating the cost function and setting its derivative with respect to the retransmission time τ to zero, given that the network does not charge for messages sent ($M_r(y,p)=0$).

2.1. No lost replies. No message transmission time.

Momentarily assume that the RPC's result message is never lost. Each transmission sent and lost before the server receives the RPC request contributes kernel cost K_r and time τ to the cost function L. Since the requestor continues retransmitting the RPC request every τ seconds until it receives the results, it retransmits $\lfloor t/\tau \rfloor$ times during the service time interval t. The initial request and the one (and only) reply contribute $2K_r$.

$$L = \sum_{k=1}^{\infty} p^k q\,(K_r+\tau)\,k + 2K_r + \int_0^{\infty} f(t)\left[t+\left\lfloor\frac{t}{\tau}\right\rfloor K_r\right]\,dt\,.$$

Recall the expected value of a geometric series.

$$\sum_{k=1}^{\infty} q\,k\,p^{k-1} = \frac{1}{q}.$$

We integrate and replace the first part of the integrand with the expected service time $\bar{x}$. We replace the geometric series with its value p/q, and rewrite the floor as a summation.

$$L = \frac{p}{q}(K_r+\tau) + \bar{x} + 2K_r + K_r \sum_{j=1}^{\infty} \int_{j\tau}^{(j+1)\tau} j\,f(t)\,dt\,.$$

We minimize the total cost L by setting its derivative with respect to τ to zero.

$$\frac{dL}{d\tau} = \frac{p}{q} + \frac{d}{d\tau} K_r \sum_{j=1}^{\infty} \int_{j\tau}^{(j+1)\tau} j\,f(t)\,dt = 0.$$

We apply the chain rule and discover that derivative of the summation telescopes.

$$\frac{d}{d\tau} K_r \sum_{j=1}^{\infty} \int_{j\tau}^{(j+1)\tau} j\,f(t)\,dt = -K_r \sum_{j=1}^{\infty} j\,f(j\tau)\,.$$

When the RPC service time is distributed exponentially, then this is a geometric series and is easily evaluated.

$$\sum_{j=1}^{\infty} j\,f(j\tau) = \frac{1}{\bar{x}} \sum_{j=1}^{\infty} j\,e^{-j\tau/\bar{x}} = \frac{e^{-\tau/\bar{x}}}{\bar{x}\,(1-e^{-\tau/\bar{x}})^2}\,.$$

The optimum timeout satisfies a quadratic equation in $e^{-\tau/\bar{x}}$.

$$\frac{p}{q} - \frac{K_r}{\bar{x}} \frac{e^{-\tau/\bar{x}}}{(1-e^{-\tau/\bar{x}})^2} = 0.$$

We find the roots of this quadratic equation by employing the quadratic formula.

$$e^{-\tau/\bar{x}} = \left[\frac{q\,K_r}{p\,2\bar{x}} + 1\right] \pm \left\{\left[\frac{q\,K_r}{p\,2\bar{x}} + 1\right]^2 - 1\right\}^{1/2}$$

Since the optimal timeout must be positive, we employ the smaller root.

$$\tau_{opt} = -\bar{x} \ln \left[\left\{\frac{q\,K_r}{p\,2\bar{x}} + 1\right\} - \left\{\left[\frac{q\,K_r}{p\,2\bar{x}} + 1\right]^2 - 1\right\}^{1/2}\right]$$

Unfortunately, our precision has left us with a clumsy, albeit exact, expression. As we are seeking something that system programmers can exploit, we approximate L by ignoring the floor within the integral. This approximation's accuracy decreases as τ grows larger than $\bar{x}$.

$$L \approx \frac{p}{q}(K_r + \tau) + 2K_r + \bar{x}\,(1 + \frac{K_r}{\tau}).$$

We set the derivative of this expression to zero or, alternatively, apply the approximation $\ln(1-x) \rightarrow -x$ to the equation for τ_{opt} above, and obtain

$$\tau_{opt} \approx q\sqrt{\frac{K_r\,\bar{x}}{pq}}\ .$$

This timeout is only accurate on local networks where the transmission time is negligible and where RPC requests are lost primarily because the RPC server is overloaded. In the remainder of this section we include lost responses and network transmission time to the model. Most of the hard work is over.

2.2. No network transmission time

We now modify our cost function to account for lost replies. Ignore message transmission time for a moment longer.

$$L = \frac{p}{q}(K_r + \tau) + 2K_r + q\int_0^\infty f(t)\left[t + \left\lfloor\frac{t}{\tau}\right\rfloor K_r\right] dt$$

$$+p\int_0^\infty f(t)\left\lfloor\frac{t}{\tau}\right\rfloor\left[\tau + K_r\right] dt.$$

$$+p\sum_{j=1}^{\infty} j\,((1+q)K_r + \tau)\,(1-q^2)^{j-1}\,q^2$$

The server's result transmission is delivered with probability q and is lost with probability p. The first integral accounts for latency and retransmission costs when the result is delivered successfully. The second integral accounts for latency and retransmission costs through the instant when the result transmission is lost, in which case the latency is a multiple of τ. The geometric series in q^2 accounts for the kernel costs and latency that accrue after the result transmission is lost until both a retransmission and the reply to the retransmission are successfully delivered. The factor of $(1+q)$ corresponds to the expected number of transmissions for each retransmission cycle. With probability 1 the requestor retransmits. With probability q the retransmission reaches the server and the server replies. (We assume the server holds the results of the RPC for a long time; clearly, some form of garbage collection can eliminate old results). We collapse the geometric series and replace the floor with a summation.

$$L = \frac{p}{q}(K_r + \tau) + 2K_r + p\,((1+q)K_r + \tau)\,q^{-2}$$
$$+ q\,\bar{x} + (K_r + p\,\tau)\sum_{j=1}^{\infty}\int_{j\tau}^{(j+1)\tau} j\,f(t)\,dt\,.$$

We minimize this quantity by setting its derivative to zero.

$$\frac{dL}{d\tau} = \frac{p}{q} + p\,q^{-2} - (K_r + p\,\tau)\sum_{j=1}^{\infty} j\,f(j\tau)$$
$$+ p\sum_{j=1}^{\infty}\int_{j\tau}^{(j+1)\tau} j\,f(t)\,dt = 0.$$

When f(t) is exponential, we easily find the cost function L.

$$L = \frac{p}{q}(K_r + \tau) + 2K_r + p\,((1+q)K_r + \tau)\,q^{-2}$$
$$+ q\,\bar{x} + (K_r + p\,\tau)\,\frac{e^{-\tau/\bar{x}}}{1 - e^{-\tau/\bar{x}}}.$$

However, this expression is not in quadratic form and we can not find τ_{opt} in closed form. Although we could apply the Newton-Raphson technique to approximate this expression's root, we choose instead to approximate τ_{opt} by ignoring the floor.

$$L \approx \frac{p}{q}(K_r + \tau) + \bar{x} + 2K_r + p\,((1+q)K_r + \tau)\,q^{-2}$$
$$+ K_r\,\bar{x}\,\tau^{-1}.$$

We set its derivative to zero and solve for τ_{opt}

$$\tau_{opt} \approx q\sqrt{\frac{K_r\,\bar{x}}{(1-q^2)}}$$

Let us investigate our approximation's sensitivity to the service time distribution. In Figure 2 we plot the cost function L and our approximation of τ_{opt}. We plot the cost function L for uniform, exponential, and constant service time distributions of equal expected service time $\bar{x}$. We see that our approximation of τ_{opt} is very close indeed.

2.3. Message transmission time

We are now ready to account for message transmission time by adding to each message its expected message transmission time y. We proceed in the familiar manner to derive the optimal timeout τ_{opt} that we seek.

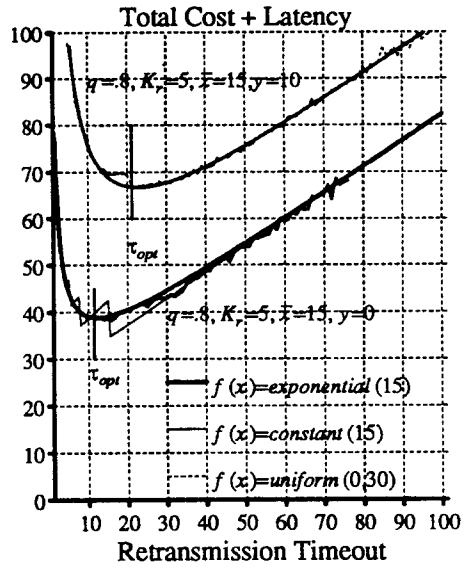

Figure 2. Figure 2. Cost function versus timeout. $\bar{x} = 15$, $K_r = 5$, $q = .8$, y=(0 and 10).

$$L = \frac{p\tau}{q} + q\,(2y + \bar{x})$$

$$+ p\left[\tau \int_0^\infty f(t)\left\lfloor \frac{y+t}{\tau} \right\rfloor dt + 2y + \frac{\tau}{q^2}\right]$$

$$+ K_r\left[\frac{p}{q} + (1+q)\int_0^\infty f(t)\left\lfloor \frac{2y+t}{\tau} \right\rfloor dt\right.$$

$$\left. + 2 - q\int_0^\infty f(t)\left\lfloor \frac{t}{\tau} \right\rfloor dt + \frac{p\,(1+q)}{q^2}\right]$$

When the server's service time is exponentially distributed, we can evaluate the cost function. However, as this does not lead to a closed form expression for τ_{opt}, we immediately derive an approximation for L by ignoring the floors.

$$L \approx \frac{p\tau}{q} + q\,(2y + \bar{x}) + p\left[y + \bar{x} + 2y + \frac{\tau}{q^2}\right] \qquad (2)$$

$$+ K_r\left[\frac{p}{q} + \frac{(1+q)\,2y}{\tau} + 2 + \frac{\bar{x}}{\tau} + \frac{p\,(1+q)}{q^2}\right].$$

We set the derivative with respect to τ to zero, replace p by $1-q$, and solve for the optimum timeout τ.

$$\tau_{opt} \approx q\sqrt{\frac{K_r\,((1+q)\,2y + \bar{x})}{(1-q^2)}}. \qquad (3)$$

In Figures 4 and 5 we plot the cost function and its standard deviation for several sets of parameters. Note that the standard deviation grows quickly when the timeout exceeds τ_{opt} and that our approximation for τ_{opt} holds for both constant and exponential message transmission delay.

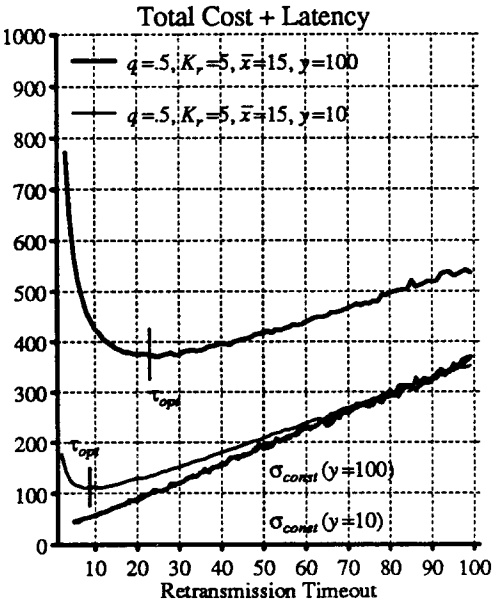

Figure 3. Figure 3. Cost function versus timeout. $\bar{x} = 15$, $K_r = 5$, $q = .5$, $y = 10$ and $y = 100$, service exponential, transmission time constant.

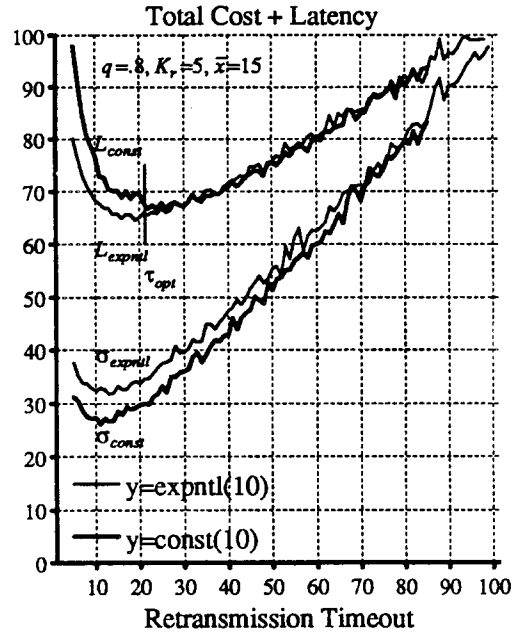

Figure 4. Figure 4. Mean and standard deviation of cost function versus timeout. $\bar{x} = 15$, $K_r = 5$, $q = .8$, $y = 10$, service exponential, transmission time constant and exponential.

3. Paying for faster messages

Since public utility provided digital communication will eventually become popular, let us investigate how one possible feature of future networks, being able to pay for reduced transmission delay and lower error rates, affects our timeout calculations. Suppose the network charges us for each message we send, charging more for decreased transmission time and lower loss rates. We, as network subscribers, specify the type of service we desire, and pay a service charge based on the network resources we consume [2]. For example, we can specify lossy service and pay less per message, but have to retransmit more. For our purposes, assume we can specify the message transmission delay y, $y \geq y_0$, and loss probability p, $p \geq p_0$, and pay a charge $M_r(y,p) \geq 0$, decreasing in both y and p, for each message we send (y_0 is the minimum network delay and p_0 is the minimum message loss probability). We find the optimal retransmission time and service grade ($\tau_{opt}, y_{opt}, p_{opt}$) subject to the constraints $\tau > 0$, $y \geq y_0$, and $p \geq p_0$ that minimize our RPC cost function.

Recall the definition of the RPC cost function, equation (1). Assume again that the scaling factors a, b, and c have already been included in $M_r(y,p)$ and K_r. We approximate the RPC cost function by combining equations (1) and (2).

$$L \approx \frac{p\tau}{q} + q\,(2y + \bar{x}) + p\left[\, y + \bar{x} + 2y + \frac{\tau}{q^2}\right]$$

$$+ (K_r + M_r(y,p))\left[\, \frac{(1+q)\,2y + \bar{x}}{\tau} + 2 + \frac{p\,(1+2q)}{q^2}\right].$$

We employ the Kuhn-Tucker method [3] of constrained optimization to minimize L. The Kuhn-Tucker method optimizes L by setting its partial derivatives to zero while maintaining all of the constraints. We make this example concrete by considering a possible message cost $M_r(y,p)$ and constraint g_p.

$$M_r(y,p) = \frac{\alpha}{y}$$

$$g_p = p - p_0 = 0.$$

$$g_y = y - y_0 \geq 0.$$

We introduce Lagrange multiplier λ_y corresponding to constraint equation g_y. The Kuhn-Tucker conditions reduce to

$$\frac{\partial}{\partial \tau} L = 0$$

$$\frac{\partial}{\partial y}\left[\, L - \lambda_y\, g_y\right] = 0$$

$$\lambda_y\, g_y = 0$$

$$\tau \geq 0, \text{ and } y \geq 0.$$

This constrained optimization problem is easily solved since p is given and the constraint on y is linear. We discover that the optimal pair (τ_{opt}, y_{opt}) is simply the solution to the unconstrained problem unless $y_{opt} < y_0$, in which case the optimal transmission time y is y_0 and the optimal retransmission timeout is τ_{opt} as calculated in (3) after substituting $M_r(y_0,p) + K_r$ for K_r and y_{opt} for y.

$$\tau_{opt} \approx q\sqrt{\frac{(K_r + M_r(y_0,p))\,((1+q)\,2y_0 + \bar{x})}{(1-q^2)}}.$$

In Figure 5 we plot several optimal timeout and optimal transmission delays as a function of the constant

of proportionality α. When $M_r(y,p)$ is differentiable and decreasing in y and p, we can always apply the Kuhn-Tucker algorithm to find the channel parameters y_{opt} and p_{opt} and retransmission time τ_{opt} that minimize L. Compare the two darkest pairs of lines which only differ by the kernel cost K_r. The darkest line has larger K_r. We see that this pushes the retransmission time higher but lets us pay for faster channel delay.

4. Implementation

We can take advantage of these timeout calculations in real distributed systems by incorporating them into network code. We can cache estimates of the expected service time $\bar{x}$, the loss parameter p, and the round trip message delay $2y$ to various destinations. We can update a destination's estimates every time we complete an RPC. Where no estimates exist, we can initialize them to some set of default values. Estimates for sites with which we communicate frequently, nameservers, authentication servers, page servers, will improve and converge.

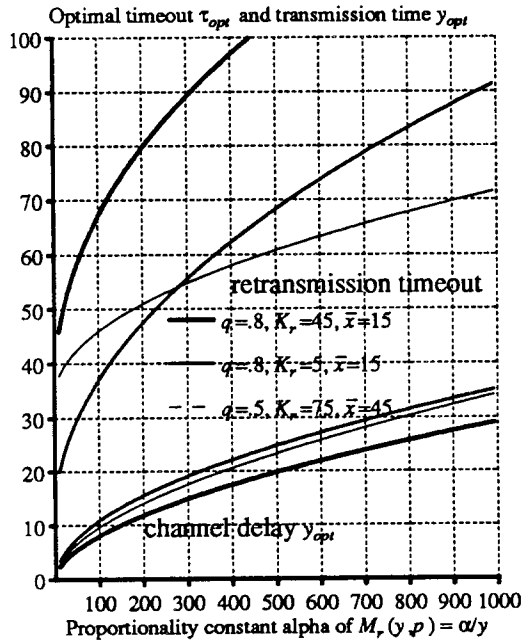

Figure 5. Figure 5. Retransmission timeout τ_{opt} and transmission delay y_{opt} are functions of the transmission delay cost function $M_r(y,p) = \alpha/y$.

It is more difficult to exploit Section 3's results, finding both the network's optimal grade of service and the optimal timeout, because we lack closed form approximations. However, we can calculate these values by solving the Kuhn-Tucker equations once and place them in the application software or operating system. Given a specific network cost function $M_r(y,p)$, we could construct a table of the optimal τ, y, and p indexed by the remaining variables $\bar{x}$ and K_r.

4.1. Stability

Address stability issue. Since losses may occur from buffer overflow, raising the timeout, not lowering it, is the measure that leads to stable behaviour. However, as congestion and losses increase, the server's estimation of $\bar{x}$ increases as well. If the server's queue holds four jobs, then its estimate of $\bar{x}$ increases to $5xbar_0$, where $\bar{x}_0$ denotes the empty queue behaviour.

When more than one service provider exists, a related problem is choosing the timeout at which we give up on one service provider and go on to the next. For example, the 4.3 bsd UNIX nameserver accepts the names of three alternative servers which it tries in round-robbin order. We can apply our technique to calculate this

timeout.

5. Conclusions

Selecting protocol timeout values taxes the intuition of system programmers. Historically RPC timeout values have been chosen long to minimally impact the network load and because the programmers assumed that losses were rare. At the minimum, our expression for the optimal retransmission timeout (3) sheds light on the intuitive process of selecting timeouts. We believe our expression replaces the black art of selecting timeouts. In the future, when we have to select and pay for the grade of network service that we use, we will have to select both the RPC retransmission time and the network transmission delay and error rate that we request. We solved this problem by adding the cost per message $M_r(y,p)$ sent into the RPC cost function L and then minimized L over the parameters that we are allowed to select, the retransmission time τ, the network transmission delay y, and the network's message loss probability p.

We believe our RPC cost function reflects the tradeoffs that protocol designers make. However, the optimal retransmission timeout that we calculate (3) is only as good, of course, as the parameters that drive it, the kernel cost per message K_r, the expected service time $\bar{x}$, the probability a message is lost p, and the round trip message delay $2y$.

$$\tau_{opt} \simeq q\sqrt{\frac{K_r\,((1+q)\,2y+\bar{x})}{(1-q^2)}}.$$

One can extend our skeletal RPC algorithm to other request-reply protocols. We can apply our techniques to optimize the Birrel and Nelson algorithm [1] in which the server acknowledges all retransmissions. Inherently, we have assumed that RPCs are not the cause of network congestion and that message loss is independent and describable by a single parameter p. Since our algorithm decreases τ with increasing loss probability p, we add to network congestion. This is reasonable since stream protocols are the principle cause of network congestion, and, in the future, charging for messages will reduce the greedy stream protocol's incentive.

References

1. A. D. Birrel and B. J. Nelson, "Implementing Remote Procedure Calls", *Trans. Computer Systems* 2, 1 (Feb. 1984), 39-59.
2. D. Anderson, "A Software Architecture for Network Communication", *8th International Conference on Distributed Computing Systems*, June 1988, 376-383.
3. M. Avriel, *Nonlinear Programming: Analysis and Methods*, Prentice Hall, Englewood Cliffs, NJ, 1976.