On the Robustness of Soft State Protocols*

John C.S. Lui
Dept. of Computer Science & Eng.
The Chinese University of Hong Kong
Shatin, N.T. Hong Kong
cslui@cse.cuhk.edu.hk

Vishal Misra
Dept. of Computer Science
Columbia University
New York, NY
misra@cs.columbia.edu

Dan Rubenstein
Dept. of Electrical Eng.
Columbia University
New York, NY
danr@ee.columbia.edu

Abstract

Soft state has been a mantra of Internet protocol design for the past decade. System designers build protocols that implement soft state mechanisms based on intuition or on qualitative arguments that the design is "better", yet there has never been a formal performance evaluation study that draws the same conclusion. In fact, previous attempts [7, 12] to build such a quantitative argument have found that pure soft state protocols significantly underperform their hard state counterparts, and that only soft-hard hybrids can match hard state protocol performance. In this paper, we argue otherwise. We develop models that provide a performance-oriented explanation and justification of the Internet designer's intuition. The novel observation is that, if network conditions are known, a hard state protocol can always be configured to outperform its soft state counterpart. However, in reality, network conditions are unpredictable, and that soft state protocols are much more resilient to unanticipated fluctuations in these conditions.

Keywords: methodology of designing network protocols, stochastic analysis, robustness.

1 Introduction

A communication protocol is a procedure used by two interacting parties to exchange information across a communication channel. The exchange of course includes the data that the receiving party wishes to obtain from the sending party. However, the same communication channel is often used by the end-points to exchange the *control* information that is necessary to successfully and efficiently complete their communication transaction. For instance, control information is exchanged for the purposes of initiating and terminating the communication session, recovering from lost transmissions, and regulating the rate of data exchange.

Protocols often maintain a modifiable *"state"* that is used to track the progress of the communication. Usually, the various endpoints' states are inter-dependent. For instance, if a connection-oriented protocol is used to transfer data from a sender to the receiver, both the sender and receiver maintain state that indicates the existence of the connection. Consistency of this state is necessary to ensure proper and efficient exchange of data: the sender should attempt to send data to the receiver when and only when the receiver expects to receive this data.

For over 15 years, the Internet community has openly encouraged the use of *"soft state"* as the means to maintain consistency between inter-dependent states. To define "soft" state, we must first define the state's *default* value. A communication endpoint's state can take on several values, but reverts to its default value if, within a given time period, the endpoint receives no communication that explicitly sets the value. This time interval is commonly referred to as the *timeout* period and its length is specified within the protocol. This returning of state to a default value is applied in almost every communication protocol in existence, regardless of whether the protocol utilizes soft or hard state design. For instance, a connection-oriented protocol can be viewed as implicitly maintaining a boolean state that is set to TRUE when the connection is open, and is FALSE when the connection is closed. The default state of the connection would be FALSE. This is because if, after a period of time, the communication endpoint receives no message within the timeout interval, it sets its state to FALSE, effectively closing the connection.

What determines the "softness" of a protocol's state is the manner in which the timeout mechanism is used to re-

[*] This material was supported in part by the National Science Foundation under Grants No. ANI-0117738, ANI-0238299, ANI-0133829, by the U.S. Department of Energy under Grant DOE DE-FG02-02ER25528, and by Cisco and the Intel Information Technology Research Council. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

vert the state back to its default value. In a hard state protocol, the timeout is used for this reversion only as an emergency failsafe, after normal operating procedures have failed. Thus, hard state protocols must implement messages whose purpose is to explicitly communicate the desire to revert state to its default value. In our example above, the normal procedure for terminating a connection in a hard state protocol would require explicit messaging that communicates that the connection should be terminated, and that the state should revert to its FALSE value. The change via timeout is used only in extreme situations when some unexpected network condition prevents this explicit messaging from completing.

In contrast, soft state protocols intentionally utilize the timeout mechanism under normal operating conditions to return state to its default value. Hence, soft state protocols do not require explicit messages that return state to its default, though such messaging may exist as an optimization.

Internet designers that promote soft state design have provided intuitive, high level qualitative explanations for why soft state protocols are "better" than their hard state alternatives [4]. However, there has never been a study that quantitatively explains this intuition from a performance standpoint. In fact, the results of previous work [7, 12] that compare the performance of hard and soft state protocols, measure the fraction of time that the communicating entities have inconsistent views of the system state. These works conclude that soft state protocols are consistent for a smaller fraction of time than their hard state counterparts. Furthermore, they posit that, in order to improve consistency, soft state protocols should incorporate a hard state-like, explicit messaging to circumvent the long lapses that may occur between the time when one communication endpoint chooses to revert to the default state and the other's refresh interval completes.

Given the above result, does this mean that Internet designers' intuition that soft state is a better design is unjustified? We show that this is not the case. *In this paper, we develop analytical models that provide an explanation that supports the Internet designer's intuition that soft state is indeed "better". Our observation is that, because networking conditions are unpredictable, it is important to construct protocols so that they are* **robust** *to the wide variety of network conditions to which they may be exposed.* By "robust", we mean that the protocol's performance under a variety of network conditions is above an acceptable threshold, but need not be optimal.

Our findings are that hard state protocols, which can be optimized to outperform their soft state counterparts under a given set of network conditions, are less robust than their soft state counterparts. As the underlying network conditions are pushed toward unexpected extremes, hard state protocol performance degrades at a much faster rate than that of soft state protocols. This observed phenomenon is in agreement with the philosophy of the mathematical theory of *Highly Optimized Tolerances* (HOT) [2], which states that design decisions that provide optimum protection against known disturbances or uncertainty can lead to catastrophic failure against unknown or rare disturbances.

In our paper, we identify three characteristics of soft state protocols that naturally make them more robust than their hard state counterparts:

1. Soft state protocols are less trusting of misbehaving endpoints.

2. Soft state protocols use the timeout mechanism to create a a virtual, predictable, feedback channel.

3. Soft state protocols are less likely to flood the network with signaling traffic when network conditions deviate from the norm.

We demonstrate each characteristic in the context of an anomalous network setting that causes the network's conditions to differ greatly from their expected norms: (1) *denial of service attacks*; (2) *overload of a shared, bidirectional communication channel*; and (3) *the rapid joining and leaving of participants from a broadcast session*. To demonstrate the first two characteristics, we use simple models of hard and soft state protocols to measure the blocking rate: the likelihood that a server is unable to serve an incoming request. The third characteristic is explored using a simple model of hard and soft state protocols that measures the feedback rate of the broadcast listeners. These three models include tunable parameters that vary the intensity of an undesirable network condition. We first set the tunable parameters to the expected intensity of the condition, and optimize the protocols' performance for this set of parameter values. We then modify these parameters to increase the intensity of the condition, and observe the performance of hard and soft state protocols.

Our results show that soft state protocols *"survive"* under more extreme conditions where their hard state counterparts fail. In essence, these results quantify that by using soft state design, the network, while perhaps often offering a slightly degraded service under normal conditions, is more robust to variations in these conditions.

The remainder of the paper proceeds as follows. In Section 2, we provide the motivation why one needs to consider robustness in the design of Internet protocols. In Section 3, we present a generic and unified model to study the performance of soft/hard state protocols and we use the denial of service attack to illustrate their respective robustness. In Section 4, we present a study of a correlated feedback lossy channel and compare the performance of soft/hard state protocols. In Section 5, we explore the implosion rate of a polling protocol and show that servers that utilize soft state protocols are not overwhelmed with messages when membership changes at a high rate. Finally, we conclude in Section 6

2 Motivation

We begin by elaborating on what we mean by "robustness", since it is this property along which we show that soft state protocols outperform their hard state counterparts. Figure 1 illustrates the general phenomenon we observe. The two increasing curves labeled A and B each plot the performance of a protocol as a function of some network condition, where a lower value indicates better performance. The performance can be one of many measures, e.g., the loss rate or delay across the channel. Similarly, the network condition can be one of many: the rate at which attackers initiate denial of service attacks, the expected lifetime of a session, or the fraction of links in a network that have failed.

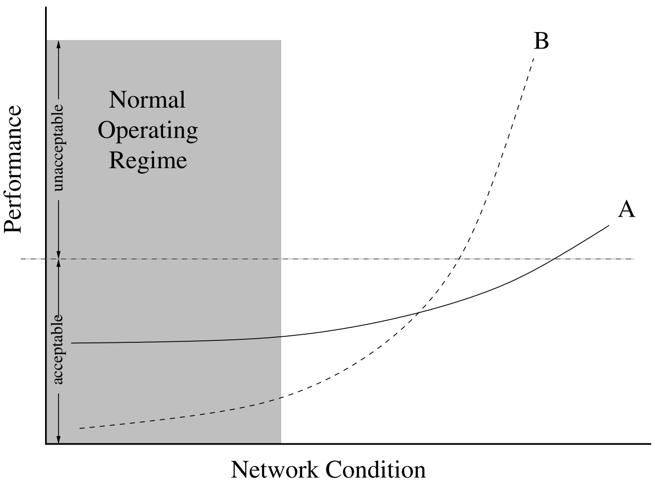


Figure 1. Robustness of a protocol.

The grey box depicts the normal operating regime of the network. On a day-to-day basis, the value of the network condition will be within the scope of the grey box, so most of the time, the performance of the protocol A, indicated by the solid curve, will be larger (worse) than the performance of protocol B, indicated by the dashed curve. The horizontal dashed line represents a threshold on the level of acceptable performance. A protocol's performance is considered to be acceptable as long as its value remains below this line, and is unacceptable when its value rises above the line. For many performance measures, determining the value of this threshold in practice is for the most part a matter of opinion or business strategy. However, for the sake of presenting our argument, assume that, in this instance, it has a fixed, known value.

Even though protocol B outperforms protocol A under normal network conditions, protocol A is more robust, in the sense that the network condition must be stretched further from its normal operating point before protocol A's performance rises above the acceptable threshold level. In the next three sections, we will construct simple yet precise models that show that when one compares hard state and soft state protocols designed for the same application, hard state protocols will mimic the behavior of protocol B and soft state protocols will mimic the behavior of protocol A.

3 The impact of the refresh timer value

In this section we perform a simple cost analysis of the design choices that determine the timer values of a soft state protocol (and the corresponding heartbeat value of hard state protocols). Before proceeding with the analysis, we enumerate the different kinds of costs that affect the operation of these signaling protocols (these costs are similar to those described in [7]).

1. Application-specific inconsistency cost.

2. State (Re)Initialization cost.

3. Refresh overhead.

4. Stale state cost.

Figure 2 presents an abstract analytical model that describes the behavior of both hard and soft state versions of protocols. The principle difference in the operation of the two protocols is the time over which the refresh timer expires, and that the soft state protocol requires a small amount of additional time after the session officially ends for the refresh timer to expire and reset the sender to its default state.

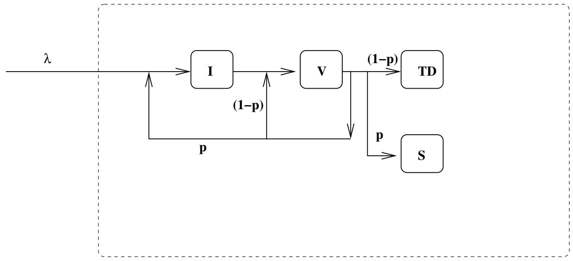


Figure 2. Analytical Model for hard and soft states.

In the model, we assume that the connection has a lifetime L (with an average connection life time of $E[L]$). The state *I* represents the initialization state. A connection is initialized via initialization messages from a sender that arrive to system at rate λ. Once initialized, the connection remains alive for L time units in the V (valid) state, with periodic refresh/heartbeats sent every R time units. Finally, when the sender decides to tear down the connection, the system transitions to the *TD* (teardown) state, or, if the teardown message is lost then it goes to the *S* (stale) state. Both refresh and teardown messages are lost with a probability p. Most implementations of protocols require the loss of a small number of consecutive refresh/heartbeat messages before state expires or times out. For simplicity of exposition we will assume that this number is 1, as it does not affect the nature of our conclusions.

The parameter in the hands of the designer is the time, R, of the refresh interval. We now explore the tradeoffs a designer faces in choosing the right R, and we measure performance by associating a different cost $C_{\cdot}(\cdot)$ with each event that transpires within the communication system. In general, these costs can be any non-linear, increasing functions of their respective arguments, but again, for simplicity of analysis we will model each cost by a linear function, i.e., the costs $C_{\cdot}(\cdot)$ are simply constants multiplying the corresponding arguments. Since the lifetime of a state is L, the total number of refresh events that happen during a connection lifetime is approximately L/R. Let the cost of each refresh message be C_r.

The protocol enters an inconsistent state when a refresh message is lost. For small values of channel loss probability, the number of such events can be approximated by pL/R. The cost of being in an inconsistent state is application specific, and is usually in proportion to the length of time spent in the inconsistent state. This induces a tradeoff on the time of the refresh timer. A shorter time increases the number of refresh messages during the lifetime of the connection and hence increases the likelihood that the system enters an inconsistent state (modulated with the channel loss probability p). Conversely, a longer time increases the expected time spent in the inconsistent state once entered.

If we let C_{is} be the cost per unit time of being in an inconsistent state, then the total cost over the lifetime of a connection is approximately $C_{is}pL$. Once the system enters an inconsistent state, there is a cost to re-initialize the connection, for instance due to a reallocation of resources at the server. The total number of such events over the lifetime of the connection is 1+pL/R (the initial installation and subsequent re-installation with every refresh message loss event). If we assign a cost of C_i for (re)initialization, then the total cost is $C_i(1+pL/R)$. The cost of a stale state is proportional to the length of time spent in the stale state, and thus is proportional to the timeout interval, which in turn is of the order of the refresh interval. Again, assigning a cost C_{ss} as the cost per unit time being in the stale state, the total cost for being in the stale state is $C_{ss}R$. For a pure soft state protocol, this cost is incurred *every time* a state expires, whereas for a hard state protocol it is incurred when the state removal message gets lost, which in turn occurs with the channel loss probability p. We assign a probability to this event p_{ss}, which is set to 1 for a pure soft state protocol and is set to p for a hard state protocol or a soft state hybrid that includes explicit state removal [7]. Now we look at the total cost as a function of the refresh interval, $C(R)$. $C(R)$ is given by

$$C(R) \;=\; C_r\frac{L}{R} + C_{is}pL + C_i(1+p\frac{L}{R}) + C_{ss}p_{ss}R.$$

The expected cost is then given by

$$\begin{aligned} E[C(R)] \;&=\; E[C_r\frac{L}{R}] + E[C_{is}pL] + E[C_i(1+p\frac{L}{R})] \\ &\quad +E[C_{ss}p_{ss}R] \\ &=\; C_r\frac{E[L]}{R} + C_{is}pE[L] + C_i(1+p\frac{E[L]}{R}) + \\ &\quad C_{ss}p_{ss}R. \end{aligned}$$

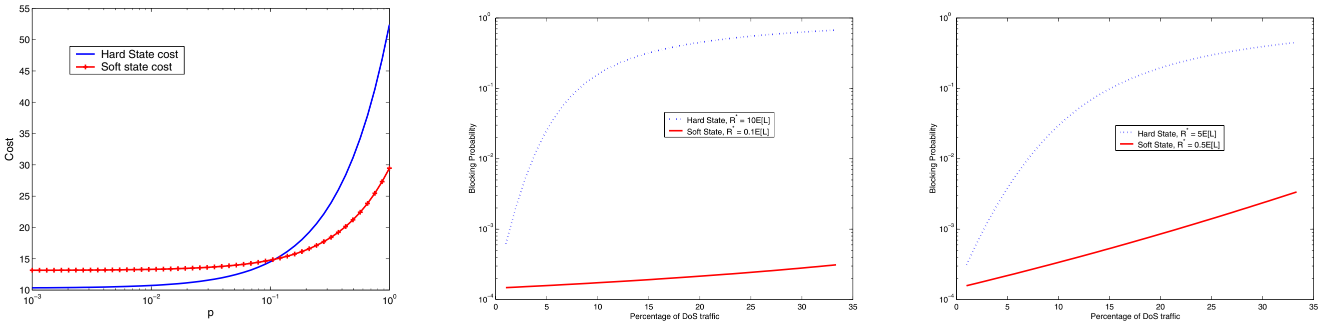
Observe that some of the costs are inversely proportional to R, whereas others are directly proportional to R or independent of R. We group together costs that are directly proportional to R and those that are inversely proportional to R, and scale those costs by constants a and b. This results in a cost equation with different emphasis on the different costs. The rewritten cost equation is then given by

$$\begin{aligned} E[C(R)] \;&=\; a(C_r\frac{E[L]}{R} + C_i(1+p\frac{E[L]}{R}) + C_{is}pE[L] \\ &\quad +b(C_{ss}p_{ss}R). \end{aligned}$$

The above equation can be solved to obtain the optimal value R^*, that minimizes the cost.

$$R^* = \sqrt{\frac{aE[L](C_r + C_ip)}{bC_{ss}p_{ss}}}. \qquad (1)$$

The formula makes intuitive sense: the longer the session lifetime, the longer one should make the timeout interval. The channel loss probability appears in both the numerator and denominator as it affects two kinds of costs, the fixed stale state cost after teardown, and the periodic state initialization costs. The crucial observation is to note that the optimal refresh interval setting is proportional to $\sqrt{a/b}$: this is the *fundamental* hard versus soft state tradeoff. A protocol (or a state) becomes *harder* with increasing values of $\sqrt{a/b}$, and, from a performance standpoint, the range of values of (a/b) define a spectrum of soft to hard state protocols. Hard



(a) Cost comparison of soft state timer vs. hard state timer as channel loss probability increases

(b) Blocking probability as a function of percentage of attack traffic. $\frac{R^*_{hs}}{R^*_{ss}} = 100$

(c) Blocking probability as a function of percentage of attack traffic. $\frac{R^*_{hs}}{R^*_{ss}} = 10$

Figure 3.

state protocols are used when the cost of concern is the refresh overhead, as well as the cost of accidentally losing an installed state, and increasing a yields a high value of R^*.

Increasing b places a greater premium on the stale state cost, and it increases robustness to unforeseen circumstances. It can be viewed as the cost of trusting a particular state, where a higher value of b results in a lower level of trust. By roughly following the procedure outlined above, a protocol designer can determine the optimal refresh interval, *given assumptions of other parameters*, i.e., assume some p, some $E[L]$, *choose* a, b to put different emphasis on different costs to determine the optimum R^*.

Let us consider an example consisting of two cases, where a protocol designer chooses the refresh interval and various costs such that $a \gg b$ for the first case and such that $b \gg a$ for the second. In the former case, the designer produces a "hard state" protocol, whereas in the latter case it is a soft state protocol. Both designs assume some fixed value of $E[L]$ and a fixed, low value of channel loss probability p. In Figure 3(a) we show how the channel loss probability impacts both soft and hard state costs. We observe that, under normal conditions (i.e., the conditions under the design assumptions), the cost of the hard state version is lower than the soft state version. As conditions deteriorate and deviate from the expected, the cost of the hard state version grows at a much faster rate than the soft state version until its cost rises above the soft state cost. The growth in cost of the soft state version is much slower, consistent with the notion we described in the previous section, of extending the acceptable operating range further as network conditions deteriorate. Similar phenomena can be observed if the expected state lifetime, $E[L]$ deviates from the one designed for.

In the next section, we see a more concrete manifestation of the benefits of having a shorter refresh timer value. We also use a more realistic non-linear cost function of the refresh timer, namely, the blocking probability of the server.

3.1 A DoS attack

One of the main consequences of having a shorter refresh timer is it enables a faster recovery from network failure, and another subtle effect is resistance to malicious behavior.

Consider a scenario where the protocol operates to reserve a resource at a server. The resource could be, for instance, a forked web server process or a socket or bandwidth allocation for a multicast channel. The TCP-SYN attacks a few years ago [3] demonstrated that an attacker could launch an effective Denial of Service (DoS) attack by reserving a resource, and then exiting without explicitly tearing down the connection. In our model above, such a DoS attack is equivalent to initiating a session with a lifetime of 0, and a channel loss probability of 1. The entire cost incurred is in keeping orphaned states alive.

Assume that the server has a finite number, N, of resources. Expecting well behaved users and ambient traffic, designers would configure N to give a low blocking probability. Without loss of generality, we assume the arrival rate of valid connections is 1 per time unit, and the mean session length is 1 time unit. Using the Erlang-B formula [9], to provide a blocking probability of about 10^{-4}, the server needs to allocate 32 resources. Let us examine, however, what happens to the blocking probabilities at two servers that are identical except that one implements a soft state connection-oriented protocol while the other implements a hard state version when an attacker starts injecting malicious connections. We plot the blocking probability of the server under attack in two scenarios, one where the ratio of the hard state refresh timer (R^*_{hs}) to the soft state refresh timer (R^*_{ss}) is 100, and the other where the ratio is 10. We gradually increase the percentage of attack traffic (in terms of arrival rate λ of malicious connections) to total traffic.

We observe in Figure 3(b) that even a modest (10 %) amount of DoS traffic can cause unacceptable levels of blocking when using a hard state protocol (i.e., close to 0.3), whereas the blocking probability does not exceed 10^{-3} for the soft state version, even when the attack traffic is as much as 30% of the total traffic. In Figure 3(c), where the ratio of the refresh timers is much smaller, 10, reveals similar behavior, where the blocking probability for the "softer" version of the protocol grows, but at a much slower rate compared to the hard state version. This demonstrates that the "softer" a protocol is, the more robust it is to attacks.

4 A Correlated, Lossy Feedback Channel

One of the obvious benefits of soft state design is that the state is naturally refreshed and returned to a consistent state simply by waiting, without requiring the receiver to contact the sender. Such a mechanism is of great importance in a network that offers "best-effort" service, since there is no guarantee that explicit communication attempting to revert the system back to its default state will succeed. In fact, there is evidence that a poorly designed protocol can be self-defeating, i.e., it behaves in a manner that worsens the network conditions and its reaction to these worsening conditions only further deteriorates the conditions. The canonical example is the phenomenon of congestion collapse: if a reliable transfer protocol attempts to keep the rate of reliable transfer fixed by increasing its rate proportional to the rate at which packets are lost, the throughput across a fixed-capacity channel will ground to a halt.

In this section, we construct a model that shows that a hard state protocol can induce a similar phenomenon. We look at a communication crossing a symmetric, lossy communication channel: heavy traffic from the server to the clients induces loss on the channel that is used by the clients to contact the server. Such a phenomenon is common, and will likely occur if the paths in the two directions share a common medium or are transmitted through the same device.

Our model contains a server that transmits fixed-rate sessions (e.g., smoothed video) and has sufficient processing power to simultaneously host up to $N > 0$ such sessions. The channel used to communicate between the sender and receiver has the capacity to support $M < N$ sessions without loss. If the number of sessions, k, is higher than M, then the channel exhibits a loss rate. Formally,

$$\text{loss rate} = \begin{cases} 0 & \text{if } k \leq M, \\ 1 - \frac{M}{k} & \text{if } M < k \leq N. \end{cases} \qquad (2)$$

We assume that the server has a backlog of session requests to serve and can initiate new connections with an average rate of λ. The mean time needed to complete the transfer of data is $1/\mu$, where the decision to terminate the connection is left to the client.

In the hard state protocol, the client attempts n times to deliver a message to the server to terminate the connection. If these explicit requests fail to reach the server, the client aborts, and the server continues to transmit session data across the communication channel. Without an appropriate failsafe, such a system is doomed to collapse. We therefore assume that the protocol includes a soft state failsafe: a time with mean $1/\mu_t$ after the time client aborts, the server can "sense" the absence of the client and terminates the session. Because, under *normal* network operations, it is unlikely that a client will fail to contact the server and terminate the connection, to prevent accidental premature terminations, $1/\mu_t$ is kept large.

In the soft state protocol, the client pings the server periodically at a high rate, and ceases to ping the server when it wishes to end the session. When the server does not receive pings for a time with mean $1/\mu_s \ll 1/\mu_t$, it terminates the session from its end.

CTMC for hard state protocol: To construct a model for above mentioned application with a tractable solution, we employ continuous time Markovian chains (CTMC). To model the hard state protocol, let $\mathcal{M}_h$ be a two-dimensional CTMC where $\mathcal{M}_h = \{(i,j) | 0 \leq (i+j) \leq N\}$. For a given state (i,j), i represents the number of *active* sessions whose clients are still interested in receiving data from the sender, and j represents the number of *inactive* (or aborted) sessions that continue to utilize the channel, even though no client receives the transmission. Session arrivals are described by a Poisson process with an average rate of λ, and active session lifetimes and inconsistency periods are exponentially distributed. Letting $p_{(i,j)}$ represents the loss probability of the communication channel when it is in state (i,j), we have

$$p_{(i,j)} = \begin{cases} 0 & \text{if } 0 \leq (i+j) \leq M, \\ 1 - \frac{M}{(i+j)} & \text{if } M < (i+j) \leq N. \end{cases} \qquad (3)$$

Let $x_{(i,j)}$ be the probability that, in the hard state protocol, the receiver fails to explicitly inform the sender of its departure:

$$x_{(i,j)} = p^n_{(i,j)}. \qquad (4)$$

In the event of such a failure, an active session becomes an inactive session. The additional time required to tear down this inactive session is an exponentially distributed random variable with mean $1/\mu_t$. Q_h, the transition rate matrix of $\mathcal{M}_h$ is

$$(i,j) \longrightarrow (i+1,j) \quad \text{with } \lambda \text{ when } (i+j) < N,$$

$$
\begin{aligned}
(i,j) &\longrightarrow (i-1,j) \quad \text{with } i\mu(1-x_{(i,j)}) \\
&\qquad\qquad\qquad\qquad \text{when } i>0, \\
(i,j) &\longrightarrow (i-1,j+1) \quad \text{with } i\mu x_{(i,j)} \text{ when} \\
&\qquad\qquad\qquad (i>0)\ \&\ (i+j)<N, \\
(i,j) &\longrightarrow (i,j-1) \quad \text{with } j\mu_t \text{ when } j>0.
\end{aligned}
$$

CTMC for soft state protocol: For the soft state protocol, let $\mathcal{M}_s$ be a two-dimensional CTMC where $\mathcal{M}_s = \{(i,j)|0 \leq (i+j) \leq N\}$ and the interpretation of the state (i,j) is similar to the $\mathcal{M}_h$ of the hard state protocol. When an active session finishes, it always becomes an inactive session for a period of time that is an exponentially distributed random variable with mean $1/\mu_s$. Let Q_s be the transition rate matrix of $\mathcal{M}_s$ and it is specified as:

$$
\begin{aligned}
(i,j) &\longrightarrow (i+1,j) \quad \text{with } \lambda \text{ when } (i+j)<N, \\
(i,j) &\longrightarrow (i-1,j+1) \quad \text{with } i\mu \text{ when } (i>0) \\
&\qquad\qquad\qquad\qquad \text{and } (i+j)<N, \\
(i,j) &\longrightarrow (i,j-1) \quad \text{with } j\mu_s \text{ when } j>0.
\end{aligned}
$$

Let $\pi_h(i,j)$ and $\pi_s(i,j)$ respectively denote the steady state probabilities of being in states (i,j) in $\mathcal{M}_h$ and $\mathcal{M}_s$ respectively. The steady state probability vector is easily computed using standard numerical methods [14].

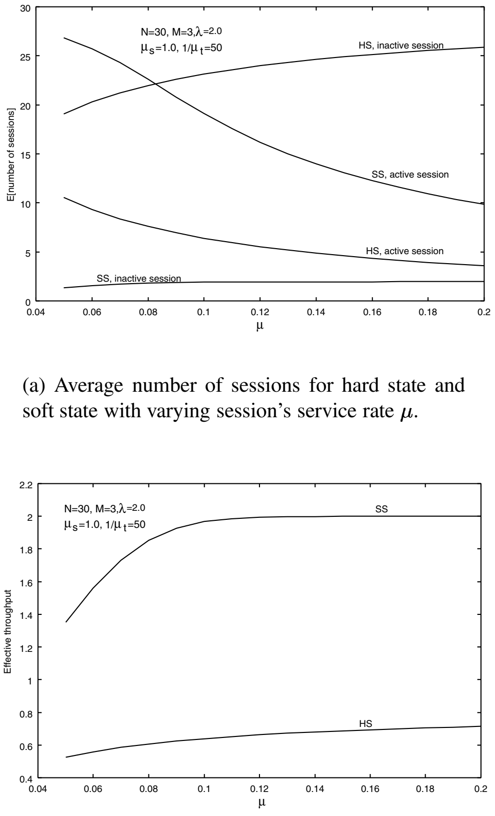
Let $S_s(t), \bar{S}_s(t)$, and $\rho_s(t)$ respectively denote the number of active sessions, the number of inactive sessions, and the effective throughput at time t of the soft state protocol. Let $S_h(t), \bar{S}_h(t)$, and $\rho_h(t)$ be defined similarly for the hard state protocol. We will use the average values of these quantities as our performance measures, which are easily determined in terms of the steady state probabilities of the respective systems:

$$E[S_x] = \sum_{i=0}^{N}\sum_{j=0}^{N-i} i\pi_x(i,j), \quad \text{for } x \in \{h,s\}. \tag{5}$$

$$E[\bar{S}_x] = \sum_{i=0}^{N}\sum_{j=0}^{N-i} j\pi_x(i,j), \quad \text{for } x \in \{h,s\}. \tag{6}$$

$$\rho_x = \lambda\sum_{i=0}^{N}\sum_{j=0}^{N-i} \mathbf{1}\{(i+j)<N\}\pi_x(i,j), \quad \text{for } x \in \{h,s\}. \tag{7}$$

Figure 4(a) to 4(b) illustrate the effect of the session's holding time $(1/\mu)$ on the three performance measures. For these figures, the system parameters are $N = 30$, $M = 3$, $\mu_s = 1.0$, $1/\mu_t = 50$ and $\lambda = 2.0$. We observe that the soft state protocol maintains a low number of inactive sessions for all values of μ and a reasonably high number of active sessions, even when the session holding time is short (e.g., $1/\mu = 5.0$). On the other hand, inactive sessions dominate the channel when a hard state protocol is utilized, significantly lowering the effective throughput.



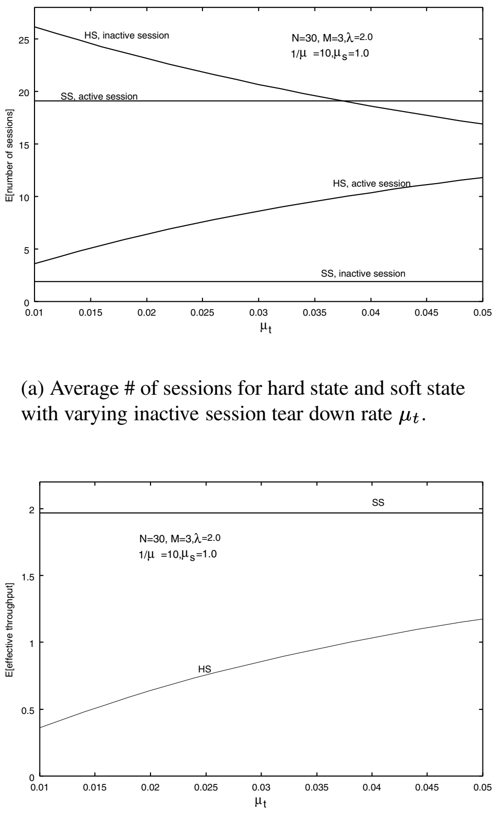
(a) Average number of sessions for hard state and soft state with varying session's service rate μ.

(b) Average effective throughput for hard state and soft state with varying session's service rate μ.

Figure 4.

Figures 5(a) and 5(b) depict the hard state protocol's sensitivity to its tear down time, $(1/\mu_t)$. Figure 5(a) shows that, as long as the tear down time is no less than the session's holding time $1/\mu$, the average number of active sessions for the hard state protocol is relatively low in comparison to that for the soft state protocol (Fig. 5(a)). Also, Figure 5(b) shows that the effective throughput of the soft state protocol is much higher than the hard state protocol for the same operating region of μ_t.

In summary, our results demonstrate that soft state protocols are more *robust* to increases in demand over a communication channel in which quality of the channel deteriorates with increasing load. In these overloading conditions, hard state protocols competing for this limited bandwidth will waste much of the bandwidth on sessions that cannot quickly be terminated.



(a) Average # of sessions for hard state and soft state with varying inactive session tear down rate μ_t.

(b) Average effective throughput for hard state and soft state with varying inactive session tear down rate μ_t.

Figure 5.

5 Broadcast Flooding

Implosion is the phenomenon where a single communication point is overwhelmed with messages that originate from a diverse set of communication points. The canonical example of implosion often used is a poorly designed reliable multicast protocol. If the protocol requires the server to receive acknowledgments from each connected client, the server's processing capabilities become the bottleneck when the number of clients grows excessively large [15]. One solution involves modification of multicast routers so that they can filter redundant messages, but this places a significant and undesirable burden on these routers. A variety of methods have been introduced in the reliable multicast context to reduce feedback levels, including timer-based approaches [5], the use of multiple multicast groups [8], and the use of parity encoding techniques [11].

In this section, we explore how soft state mechanisms can be (and have been) used to reduce client feedback that, under extreme conditions, would otherwise cause implosion. These methods have been used extensively in the reliable multicast community to minimize levels of implosion within reliable multicast communication [6, 13]. The scalability of the mechanism arises in many-to-one communication scenarios where the consistent state of the server is the 1-bit logical OR of the receiver's 1-bit states. In other words, we consider scenarios where the server's state should be in the default state (state "0") when and only when all receiver's states reside in the default state. For instance, such a soft state mechanism is used within the IGMP protocol to identify which multicast group's transmissions should be transmitted to the LAN. Multicast traffic for a particular group G is transmitted by the router onto the LAN when there is *at least* one receiver on the LAN that wishes to receive this traffic.

When multiple clients are interested in receiving the transmissions from a particular group, rather than having each client individually contact the router with its request, the router continuously listens for requests during consecutive time intervals of length T. If any client should contact the router during a particular interval, the router forwards traffic from G onto the LAN for the remainder of that interval, as well as for the subsequent interval. If no requests are received during a given interval, then the router ceases to forward traffic from G onto the LAN until it again receives a request. Receiver requests are broadcast on the LAN, and receivers randomly choose a time within the interval to transmit to the router. A receiver that wishes to receive transmissions from G can suppress its broadcast request if it hears a similar request previously broadcast by a neighboring receiver. Hence, the expected number of transmissions to the router is reduced.

Because of propagation delays, the suppression mechanism described above limits, but does not completely prevent redundant transmissions. If the time for communications to cross the LAN is τ, the size of the router's listening intervals is T, and there are n clients who choose their transmission time uniformly within the interval $[0, T - \tau]$, then the expected number of messages that reach the server is

$$F(\tau, T, n) = 1 + \frac{\tau}{T}n - \left(\frac{\tau}{T}\right)^n. \qquad (8)$$

Note that the growth is effectively linear in the number of clients. The same holds true if the client selects its waiting time from a distribution other than uniform [10].

5.1 Simple Polling Protocols

We begin by looking at a straightforward implementation of hard and soft state join/leave protocols used to perform a polling process. We consider a server that offers a single stream for broadcast. The server performs the broadcast as long as there exists a client interested in receiving the data. When the server is broadcasting, all clients receive a copy of the broadcast, regardless of whether or not they are interested in it at that time. Clients communicate to the server via unicast; these messages are queued at the server for processing. To prevent implosion, the aggregate rate of client messages to the server must be controlled.

We assume that the process by which clients interested in the broadcast arrive to the system is Poisson with rate λ and that their interest lasts for a time exponentially distributed with rate μ. This arrival process is described by the traditional $M/M/\infty$ queueing system. We define Π_i to be the steady-state probability that there are i clients that wish to receive the broadcast, where

$$\begin{aligned} \Pi_0 &= e^{-\lambda/\mu} \\ \Pi_i &= e^{-\lambda/\mu}\frac{(\lambda/\mu)^i}{i!} \end{aligned}$$

In out initial hard state protocol, each client explicitly contacts the server upon arrival and departure to respectively register and revoke its interest in the broadcast. The server tracks the individual status of each attached client, and therefore knows precisely when there exist clients that desire the service. By doing so, it knows when clients are attached and hence knows when to broadcast. Note that each arriving client sends two control messages: a join and a leave. Hence, control messages arrive at rate 2λ, and the server is actively transmitting on the channel to some client a fraction $1 - \Pi_0$ of the time.

In our initial soft state protocol, the server uses the timer-based polling mechanism described above. In each interval of T seconds, some client interested in receiving the broadcast must contact the server to continue the broadcast in the next interval. We assume that the time taken for a client to send a transmission to a server and for the server to transmit messages to clients in response to the received transmission takes time τ. Each interested client selects a point in time chosen uniformly within each interval for which it remains interested in the broadcast. The transmission from the client can be unicast, and the server can inform other clients interested in the broadcast that it has been contacted by a client in the current interval by piggybacking the information within its broadcast. Upon receiving the piggybacked information, other interested clients can suppress their transmissions until the next interval, where the process is repeated. Clients that wish to join at a time when the server is not broadcasting send an immediate join request.

Client messages arrive at the server at rate $\sum_{i=0}^{\infty}\Pi_i F(\tau, T, i)$ (where $F()$ is defined in Equation (8)). We assume that the server always broadcasts when there exists a client in the system interested in receiving the broadcast. Hence, the fraction of time the server is broadcasting is bounded below by $1 - \Pi_0$. We compute an upper bound by noting that the server will only broadcast for the entire duration of the ith interval if some client desired transmission at the start of the $(i-1)$st interval, or some client arrived during the $(i-1)$st interval. Otherwise, the server will broadcast for at most time $T - \tau - t$ if no client is interested in receiving the transmission during the $(i - 1)$st interval and an interested client arrives at time t during interval i. This gives an upper bound of $T(1-\Pi_0)+T\Pi_0(1-e^{-\lambda T})+\Pi_0 e^{-\lambda T}\int_{t=0}^{T-\tau}(T-\tau-t)\lambda e^{-\lambda t}dt$ $= 1-\Pi_0+\Pi_0/T(T(1-e^{-\lambda T})+e^{-\lambda T}(T-\tau+e^{\lambda(\tau-T)})/\lambda-1/\lambda)$.

5.2 Robustness in the presence of many receivers

The control messaging overhead of both the hard state and soft state protocols described above grows quickly with increasing λ. Before comparing these simple polling protocols, we also introduce modified versions that reduce the number of control messages as the number of clients in the system grows large.

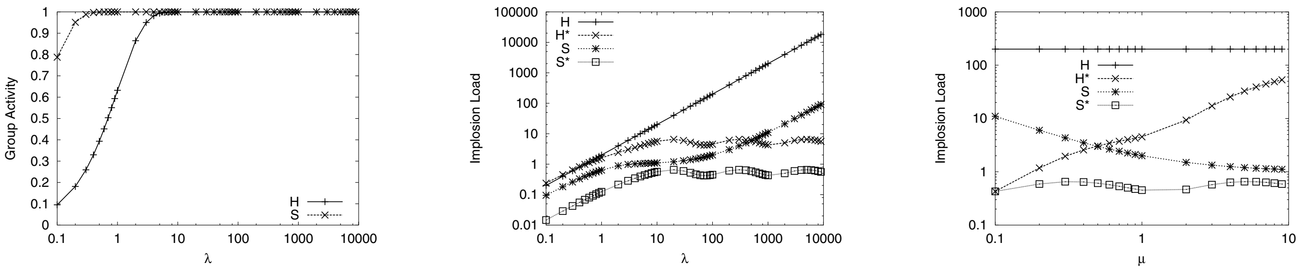
For the hard state protocol, rather than have all clients connect to the server, a leader is chosen and the server maintains the state of the leader. Only when the leader leaves is a message sent to the server, which then initiates a process to identify another leader. The server must perform a polling operation to identify a new leader. The polling operation, which we describe below, is a method for scalably identifying a new leader, and is also used by the soft state version to scalably determine if any clients are currently interested in receiving the broadcast. This mechanism is an extension of the mechanism described in [1] which was used to implement a scalable multicast feedback mechanism.

To perform this more scalable polling operation, each client is assigned a unique n-bit sequence (e.g., its IP address). The server segments its T-second polling interval into a series of $k + 1$ rounds, numbered 0 through k. It specifies an m-bit quantity where $m = ks > \log n$ for some integer s. In the ith round, a client whose unique bit-sequence (e.g., IP address) matches along the first $m - ik$ bits of the sender's specified quantity transmits a message. If any client transmits a message in round i, then no further rounds are needed - the sender has identified an active client, or, in the case of the hard state protocol, has a sampling of clients from which it can choose a leader. If no messages are transmitted by the completion of the kth round, then no clients currently desire the transmission.

The probability q that an arbitrarily chosen receiver transmits on the ith round when there are n receivers is

$$q = \begin{cases} 2^{-n} & \text{if } i = 0 \\ 2^{-k(s-i)}(1-2^{-k})(1-2^{-k(s-i-1)})^{n-1} & \text{if } i > 0. \end{cases}$$

This is the probability that a receiver transmits times prob-



(a) Fraction of time the server actively broadcasts.

(b) $\mu = 1, T = 10, \tau = 0.1$

(c) $\lambda = 100, T = 10, \tau = 0.1$

Figure 6.

ability that it did not transmit in previous rounds, given that it transmits in this round times the probability that no other receiver transmitted in the previous round.

Computing the expectation by summing over all n receivers over all rounds, we get $N(n, k, s) = n2^{-n} + n\sum_{i=1}^{s} 2^{-k(s-i)}(1 - 2^{-k})(1 - 2^{-k(s-i+1)})^{n-1}$. Substituting $ks > \log n$, we get that this quantity is smaller than $(1 - 2^{-k})\sum_{i=1}^{s} 2^{ki}e^{-2^{k(i-1)}}$ as n grows large. The second term has no dependency on n, and the first term shrinks as n grows.

The control traffic rate for the soft state protocol becomes

$$\sum_{i=0}^{\infty} \Pi_i N(i, k, s)/T.$$

For hard state, it becomes

$$\lambda \Pi_0 + \mu \sum_{i=0}^{\infty} \Pi_i N(i, k, s).$$

When there are no clients, a new leader is selected at rate λ, and when there are clients, a new leader is selected at rate μ, where traffic overhead is $N(i, k, s)$.

5.3 Analysis

Our analysis of the performance of the various client polling mechanisms assumes that the normal operating environment is one where both λ and μ are small. We fix k and s at 4 and 8 respectively. Even when the number of clients is extremely large, the expected number of transmissions received within an interval of size T is below 7. The only remaining tunable parameter is T, which we set to 10 seconds.

Figure 6(a) shows the fraction of time for which the server is actively transmitting as a function of λ when $\mu = 1$, $T = 10$, and $\tau = 0.1$ for both hard and soft state versions of the protocols. We see that for low values of λ, the hard state version spends significantly less time broadcasting traffic than the soft state version. However, as λ increases, the system quickly converges to a state where the transmission is broadcast all the time. These results show that the increased robustness gained from using soft state comes with a tradeoff when client join rates are low. In particular, when loads are low, the additional inconsistency in the soft state version causes the broadcast mechanism to be turned on a larger portion of the time. From a robustness standpoint, however, this is not a concern: the system is less efficient when loads are light.

In Figures 6(b) and 6(c), we plot the rate at which control messages arrive at the server from the collection of clients. Curves labeled 'H' and 'S' respectively plot the initial versions of client hard and soft state feedback mechanisms. The curves labeled 'H*' and 'S*' plot the respective versions with the bit-vector feedback approach. In both figures, $T = 10$ and $\tau = 0.1$. In Figure 6(b), μ is fixed at 1 and λ is increased along the x-axis. We see that, for the protocol to remain robust under high join rates, it is necessary to implement the bit-vector mechanism to reduce feedback. In Figure 6(c), λ is fixed at 100 and μ is varied along the x-axis. We see that as μ is increased, increasing the rates at which sessions leave the system, the feedback rate in the hard state protocol that implements the bit-vector mechanism grows quickly. In contrast, the soft state version maintains a low feedback rate, even for large values of μ.

We see that in extreme settings, where join and leave rates are high, a soft state protocol is needed to prevent the server from becoming overwhelmed with high rates of feedback from the rapidly joining and leaving clients. The cost of using soft state is an increased utilization of the broadcast channel under non-extreme conditions where join and leave rates are low. However, appropriately configured networks should have sufficient bandwidth available to support the additional broadcast demands imposed by using soft state protocols when the broadcast is under light demand.

6 Conclusion

In this paper we have compared the robustness of soft and hard state protocols. Rather than comparing the performance of the protocols in a known set of network conditions, we compare the performance of the protocols as we vary the network conditions from the normal operating points. Using three different scenarios, we are able to demonstrate that soft state maintain an acceptable level of performance across a much wider range of network conditions than is maintained by their hard state counterparts. This study supports and explains the Internet Designer's intuition that one should apply soft state design principles when designing protocols for the Internet.

References

[1] J. Bolot, T. Turletti, and I. Wakeman. Scalable Feedback Control for Multicast Video Distribution in the Internet. In *Proceedings of ACM SIGCOMM'94*, pages 58–67, London, U.K., August 1994.

[2] J. M. Carlson and J. Doyle. Highly optimized tolerances: A mechanims for power laws in designed systems. *Physical Review E*, 1999.

[3] CERT Advisory. http://www.cert.org/advisories/CA-1996-21.html, 1996.

[4] D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *Proceedings of SIGCOMM'89*, Austin, TX, September 1989.

[5] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, December 1997.

[6] T. Friedman and D. Towsley. Multicast Session Membership Size Estimation. In *Proceedings of IEEE INFOCOM'99*, New York, March 1999.

[7] P. Ji, Z. Ge, J. Kurose, and D. Towsley. A Comparison of Hard-state and Soft-state Signaling Protocols. In *Proceedings of SIGCOMM'03*, August 2003.

[8] S. Kasera, J. Kurose, and D. Towsley. Scalable Reliable Multicast Using Multiple Multicast Groups. In *Proceedings of ACM SIGMETRICS'97*, Seattle, WA, June 1997.

[9] L. Kleinrock. *Queueing Systems Vol I: Theory*. John Wiley & Sons, New York, 1975.

[10] J. Nonnenmacher and E. Biersack. Scalable Feedback-for Large Groups. *Transactions on Networking*, June 1999.

[11] J. Nonnenmacher, E. Biersack, and D. Towsley. Parity-Based Loss Recovery for Reliable Multicast Tranmission. *Transactions on Networking*, August 1998.

[12] S. Raman and S. McCanne. A Model, Analysis, and Protocol Framework for Soft State-based Communication. In *Proceedings of SIGCOMM'99*, Cambridge, MA, September 1999.

[13] P. Sharma, D. Estrin, S. Floyd, and V. Jacobson. Scalable Timers for Soft State Protocols. In *IEEE Infocom'97*, Kobe, Japan, April 1997.

[14] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton Press, 1994.

[15] D. Towsley, J. Kurose, and S. Pingali. A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols. *IEEE JSAC*, 15(3):398–406, April 1997.