Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Computer Networks 42 (2003) 211–229

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

Connection caching to reduce signaling loads with applications to softswitch telephony ☆

Matthew Stafford [a], Xiangying Yang [b,*], Gustavo de Veciana [b,1]

[a] *Cingular Wireless, 9505 Arboretum Boulevard, Austin, TX 78759, USA*

[b] *Department of Electrical and Computer Engineering, The University of Texas at Austin, Engineering Science Building (ENS) 516, Austin, TX 78712-1084, USA*

Received 10 May 2002; received in revised form 17 January 2003; accepted 17 January 2003

Responsible Editor: G. Kesides

Abstract

In order to support quality of service, many network operators continue to rely on connection oriented technologies, even as their networks migrate towards packet switching. Such technologies can allocate resources according to user requirements, with path stability providing low jitter for real-time services. CBR/VBR traffic classes in ATM networks fall within the connection oriented paradigm, as do Traffic Engineering initiatives such as CR-LDP and RSVP-TE protocols (from the IETF's MPLS working group). As the capacities of network links increase, network switches/routers will need to support signaling loads that grow in proportion to these increases in bandwidth. However the capacity of such elements to process connection setup and teardown requests may not grow as quickly as transmission bandwidth. The resulting congestion in the connection control plane leads to delays in connection setup (and, in extreme cases, to setup failures). In response, next-generation switch vendors have implemented various connection-caching schemes. Despite this motivation, the problem of how to design an effective caching scheme appears to be little-studied in the literature. In this paper we propose a dynamic connection caching strategy which achieves a trade-off between bandwidth utilization and decreased signaling load. The proposed scheme is based on the optimal policy for a Markov decision process; this Markov decision process models the dynamics of caching policies on a single link. We extend the single-link approach to the network context. Simulations show that the proposed mechanism is robust and effective at reducing the signaling load without significantly decreasing throughput. The simulation scenarios feature network topologies that are appropriate for softswitch telephony, thereby demonstrating the applicability of our approach in this context.

Keywords: Connection caching; Signaling; Markov decision process

☆ An early version of this work was presented at the Fifth INFORMS Telecommunications Conference, Boca Raton, FL, March 7, 2000.

* Corresponding author. Tel.: +1-512-731-0175; fax: +1-512-471-5532.

E-mail addresses: matthew.stafford@cingular.com (M. Stafford), yangxy@ece.utexas.edu (X. Yang), gustavo@ece.utexas.edu (G. de Veciana).

URL: http://www.ece.utexas.edu/~gustavo.

[1] Tel.: +1-512-471-1573; fax: +1-512-471-5532.

1389-1286/03/$ - see front matter © 2003 Elsevier Science B.V. All rights reserved.
doi:10.1016/S1389-1286(03)00190-7

1. Introduction

We study the effectiveness of connection caching as a means of reducing signaling load in telecommunication networks. *Signaling load* is our term for the tasks involved in setting up and tearing down connections. If the processing resources that perform these tasks are overloaded, deleterious effects on system performance result:

- delays in setup/teardown procedures increase to the point that industry-standard delay requirements are not met, and
- setup/teardown procedures may fail altogether.

While it eases signaling load, caching exacts a toll in terms of throughput; we wish to quantify this cost and to shed light on some interesting trade-offs.

Our model does not explicitly include connection setup delay. In view of the above observations, our model uses signaling load as a proxy for delay. Our focus is on bandwidth allocation; the notion of effective bandwidth can be used to treat variable bit rate traffic types within our modeling framework.

It is important to point out a major difference between connection caching and route caching. Although the latter addresses delay associated with route computation, it does nothing to reduce signaling overhead for setup and teardown procedures. Since our main motivation in this paper is to reduce signaling load, we cannot overemphasize this crucial distinction.

Organization of the paper. After detailing our motivation, we formulate a Markov decision process (MDP) model that describes the dynamics of a single link with a state-dependent caching policy. We discuss a numerical implementation of policy iteration; based on our results, we propose a linear dynamic caching heuristic that appears to be near-optimal for the single-link case. Finally, we validate the efficacy of this heuristic with simulation results in the single link and network contexts.

1.1. Motivation

As mentioned in the abstract, operators prefer to employ connection oriented technologies for real-time services: path stability provides low jitter as well as a familiar paradigm for operations, administration and maintenance. Note that connection caching schemes could be combined with arrangements in which there is more than one call per connection, and therefore do not preclude such multi-call arrangements. In the same vein, ATM Adaptation Layer 2 (AAL2) deployments may feature AAL2 switching/subcell multiplexing nodes which experience heavy signaling load even though switches that function only at the ATM layer are unaffected by setups and teardowns happening within the AAL2 layer.

For our purposes, a telecommunication network is a set of switches connected by transmission links. A switch has the capacity to direct traffic from any input link to any output link. A *connection* is a collection of resources (such as transmission capacity) allocated to a specific user at various points in the network.

In our model, software on the network explicitly allocates/deallocates resources as connection requests arrive, are served, and are then completed. *Signaling* is the process by which system software accomplishes this goal. *Connection setup* is the allocation process that takes place when an arriving call request is accepted. *Connection teardown* is the process of returning allocated resources to the available pool once a call has completed. A schematic representation of a switch appears in Fig. 1. The software processes that manage signaling reside in the controller.

In today's networks, 10 Gbps transmission links are coming into wide-spread use. Signaling capacity, however, has not kept the pace—for example, a capacity of 3K calls/s (setups and teardowns) is quite good for an ATM switch. A 10

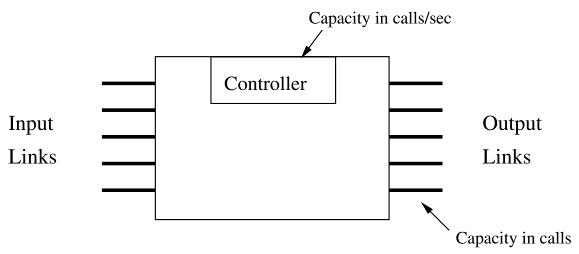

Fig. 1. Representation of a telecommunication switch.

Gbps link can carry 128K simultaneous (uncompressed) telephone conversations; if such a link is run at 80% utilization (with telephone traffic only), there will typically be around 100K calls in progress. With an average call holding time of 5 min ($= 300$ s) per call, this means that a switch will have to process about 330 call setups and teardowns per second for a *single* link. Thus a switch with a 3K calls/s signaling capacity could not support 10 links, each with 10 Gbps transmission capacity, with the offered load described above. As terabit switching fabrics become available, we expect the signaling capacity of such systems to lag behind.

Naturally, signaling capacity of switches and routers will evolve along with link speeds and switching fabric capacities. However, we see the following motivation for our view of call processing capacity as a scarce resource:

- Migration of traditional telephony services towards softswitch architectures promises to place new demands on broadband switches and routers. Control-plane interworking of softswitch components with traditional telephone equipment is complex and inevitably consumes time. Thus acceptable call setup latencies will be difficult to maintain.
- Compression schemes for delay-sensitive traffic types (such as voice and video) are improving and are becoming more widely deployed. Of course, compressed streams require less transmission bandwidth than uncompressed streams. But the per-connection signaling requirement may not decrease (indeed, it may increase because of added sophistication).
- Effective techniques for reducing signaling load could extend the useful lifetime of older switches and routers whose signaling capacity is inadequate for the emerging mix of services.

In this paper, connection *caching* will refer to delayed connection teardown: when a call completes, the connection is retained. Network resources associated with this cached connection are still reserved, but are unused. A new call request can be bound to a cached connection if (and only if)

(1) the network entry and exit points are the same as those of the initial request (i.e. the request for which the connection was originally set up),
(2) the resource requirements are the same as those of the initial request on each of the switches and links involved.

In this paper, we will assume that any two connections that traverse a given link or switch have identical resource requirements on that link or switch. Thus, in our model, requirement 1 is the only prerequisite for reuse of a cached connection.

We will refer to the reuse of a cached connection to carry a new call as a *cache hit.* Each cache hit avoids the effort of a teardown and subsequent setup procedure. Referring to the network of Fig. 2, suppose a user at switch A calls a user at switch B and the associated connection is cached when the call completes. The next time a user at A calls a user at B, the call request will be bound to the existing connection. Establishing this binding requires *some* effort on the part of switches A and B (although arguably less than that involved in teardown and subsequent setup). However, toggling an A–B connection between active and cached states involves no state change on the link connecting switches C and D and therefore places no processing load on the controllers at C and D. This is the main benefit of caching: signaling load is reduced.

The main drawback of caching is that it tends to decrease throughput. *Throughput* is the expected number of calls accepted per unit time. A

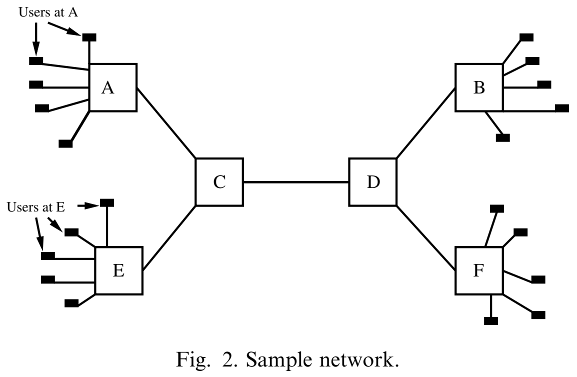

Fig. 2. Sample network.

connection request which does *not* have the same requirements as the original connection (e.g. an E–F call request in the network of Fig. 2) may be blocked because of the resources consumed by cached connections.

To summarize, the fundamental trade-off of connection caching is reduced signaling load (which is of course desirable) at the expense of reduced throughput.

1.2. Previous work and comparison with route caching and web caching

A connection is an end-to-end allotment of network resources. When a connection is cached, these resources remain reserved, and associated with one another as an end-to-end entity. Thus connection caching is very different than other caching schemes, such as route caching (which is studied in [1,8], for example). In route caching, a network device stores results of routing calculations in an internal table. Route caching schemes try to make intelligent decisions about when to (re-)compute routes and/or which routes will be discarded when storage constraints force such discards. Whether a network element keeps or discards a routing table entry has no effect on other network elements (at least, it has no direct effect). Tearing down a connection, on the other hand, typically changes the state of many network elements. Note also that route caching does nothing to reduce signaling overhead for setup and teardown procedures, but only addresses delay associated with route computation.

Our work is also significantly different from web caching. As is the case with connection caching, web caching schemes aim to reduce overhead by achieving high hit ratios. Moreover, many well known schemes provide effective caching on the World Wide Web [4]. Thus one may ask whether we can, by treating each connection as an object, apply a web caching scheme to achieve our goal of reduced signaling load. That is, based on some metric, why not bump the lowest-valued connection from the chunk of cached connections and cache a new connection with higher expected value? Web caching schemes (as exemplified by LRU and generalizations such as LFU, GD-size and Hybrid policies [3,5]) could conceivably be applied in the connection caching context. Substantial modification would be required, however.

Unmodified, the web caching replacement policies mentioned above do not reduce signaling load at all because of frequent bumping and replacing actions; this point stands out when we examine an LRU-like policy in Section 1.3. Second, current web caching schemes fail to provide intuition on how to determine the *number* of each connection type that should be cached: in web caching the cache buffer is fixed and an object will have *at most one copy in the cache*. In connection caching, the "number to cache" is critical: each additional cached connection of one type means less available bandwidth (and thus potentially higher blocking probability and signaling rate) for other types of connections. So the dynamic decision on the number to cache is one of the keys to the trade-off between blocking and signaling. Thus it is clear that connection caching and web caching indeed address different scenarios. Our simulations show that our scheme has the same desirable properties in the connection caching scenario as LFU has in web caching, even though the starting points for the two schemes are very different. Specifically, our approach favors connection types who have higher arrival rate. Moreover, our scheme could be easily modified to incorporate the consideration of size and value into connection caching, which is similar to GD-size and Hybrid's approach in web caching.

To the best of our knowledge, connection caching has received very little attention in the literature. In a recent paper [11], Serbest et al. study the performance of a connection caching scheme with adaptive timeouts.

Lippmann's 1975 paper [7] stimulated interest in the MDP framework as a tool in the analysis of queueing systems. Our analysis is similar to that supporting the use of trunk reservation in network routing mechanisms to increase revenue [6,10].

1.3. Caching policies

Having motivated the use of caching, we turn to the question of what caching policy to employ. To that end, we ask

(1) Under what circumstances should an "outgoing" connection be cached?
(2) How long should an unused cached connection be retained?

In the next two sections, we will consider general state-dependent caching policies, analyze these policies using MDP machinery, and demonstrate that there are "good" policies with some simple structural properties. Before doing so, however, we discuss candidate caching schemes and argue that our study of state-dependent caching is worthwhile.

We emphasize that none of the schemes considered here set up cached connections prior to experiencing demand. That is, the *only* sort of decision that leads to the existence of a cached connection is a decision not to tear down a connection upon completion of the associated call.

(1) *Always cache + LRU bumping of cached connections*. As the nomenclature suggests, whenever a call completes, its connection is cached. For arrivals, the "flowchart" is as follows.
    (a) An incoming call request is served using a cached connection with appropriate network entry and exit points if one is available.
    (b) Otherwise, if sufficient idle capacity is available, a connection setup is performed in order to carry the call.
    (c) Otherwise, we try to bump cached connections in least recently used (LRU) fashion to free sufficient resources for the incoming call request. If this last option fails, the call is blocked.

    Note in particular that we do not preempt active connections in this policy (or in any other policy studied in this article).

    On the surface, the "Always cache with LRU bumping" scheme has an appealing simplicity, and there is no throughput penalty under this policy. The problem with this policy is that it is ill suited to networks with distributed control. Referring to the network of Fig. 2, suppose an E–F connection request wants to "bump" an A–B cached connection. Then E must communicate its desire to A. Furthermore, A must notify E once the connection teardown procedure is complete. In a sense, this re-creates the signaling overhead that we hope to ameliorate with caching. Therefore we eliminate this approach from further consideration.
(2) *Limit cache size*. In this policy, we cache outgoing connections subject to a per origin–destination pair (per O–D pair) limit on the number of cached connections. This should limit the overall throughput penalty due to caching, while still reducing signaling load. The problem with this scheme is twofold:
    (a) One must find appropriate settings for each of the O–D pair limits.
    (b) This scheme is not responsive to changes in the distribution of traffic load among O–D pairs (unless the O–D pair limits are updated).
(3) *Timeout-only policy*. Always cache but place a timeout on each cached connection. This approach also limits the throughput penalty due to caching. We see three problems with this approach:
    (a) Finding appropriate settings for timeout(s).
    (b) Unless the timeouts are very well tuned, low-demand O–D pairs can experience disproportionately high blocking rates.
    (c) Cache sizes for individual O–D pairs are allowed to vary widely, potentially increasing blocking and reducing stability.
(4) *State-dependent caching with timeouts*. Here we make each cache/release decision based on system state. The system state will include the number of cached and active connections for each O–D pair. In a complex network, the paths followed by these existing connections may also need to be included in the state information.

    By introducing this class of policies, we are attempting to combine the virtues of policies 2 and 3 and generalize the framework for additional flexibility. The problem with this approach is that, due to the size of the state space, state-dependent policies can potentially be extremely complex. Our goal is to find simple design heuristics that produce good performance.

We are seeking an approach that is suitable for networks with distributed control and that adapts well to changing traffic loads; the first scheme has already been eliminated. We argue that timeouts are necessary to achieve our goals, and that the second candidate scheme (limit cache size) should also be removed from consideration: for example, if an O–D pair becomes inactive for a period of time, any cached connections for that O–D pair remain in place throughout that period, regardless of its duration. To address this, the per O–D pair limits would have to be frequently updated, and such a scheme would become very difficult to administer.

We will return to the timeout-only policy (third in the list) when we discuss simulation results in Section 5.

2. A single-link model

We now focus our study on the performance of caching on a single link. We abstract the network context of the previous discussions as follows: each O–D pair will be represented by a distinct stream of arriving call requests. Each of these arrival streams will be regarded, in our single-link model, as originating from a distinct user *type*. To clarify this abstraction, we return to the example network of Fig. 2. Link C-D "sees" arrival streams from four user types. This is because four O–D pairs (namely, A–B, A–F, E–B and E–F) contend for transmission capacity on link C–D; these O–D pairs map to the four user types mentioned above.

Here, then, is the general description of our model. A single transmission link having a transmission capacity of C connections is shared by N types of users. For $N = 2$, our link is schematically represented in Fig. 3. The user types differ in that a cached connection can be reused by an incoming request if and only if the user for which the connection was originally set up is of the same type as the currently requesting user. (Here we remind the reader of our assumption that each call requires the same transmission capacity on our link regardless of the user type.) Note that if there is only one user type sharing the link, cached connections are always reusable, and we see that caching does not increase blocking. Thus, in this case, connections for calls completing service should always be cached, and there is never an incentive to tear down a cached connection. This yields a regime in which the long-term average rate of call setups is 0 and throughput is constrained only by link transmission capacity. Thus $N = 2$ is the first interesting case.

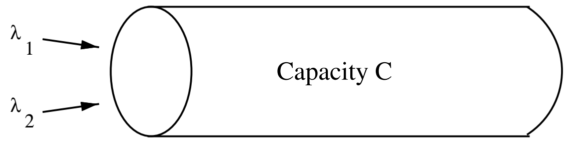

Fig. 3. Single link with resource contention.

Transmission capacity for a single connection (a.k.a. a *call*) will be called a *channel*. The arrival processes are independent Poisson processes with rate λ_n for type-n users, $n = 1, 2, \ldots, N$. Call holding times are independent identically distributed exponential random variables with rate 1. We formally define *signaling rate* to be the steady state average number of call setups/teardowns required per unit time; throughput is the steady state average number of calls accepted per unit time.

Without caching, each channel is either idle or in use by a type-n user, $n = 1, 2, \ldots, N$. Arriving requests, regardless of user type, are blocked if and only if there are no idle channels in the system. Since Poisson arrivals see time averages, all types of users experience the same blocking probability. Thus, without caching, the system is simply the $M/M/C$ queue; signaling load and throughput are easy to calculate using the Erlang-B blocking probability.

When caching is enabled, each channel can be idle, in use by a type-n user ($n = 1, 2, \ldots, N$), or cached for type-n users ($n = 1, 2, \ldots, N$). When a type-n user completes a call, the occupied channel can enter the idle state or the "cached for type-n users" state. In the latter case, an incoming call request can use that channel if and only if the request is made by a type-n user. At the time a connection enters a cached state, a timer is started; if no request by the right type arrives before the timer expires, the channel enters the idle state (i.e. the connection is torn down and the channel can be used by any user, regardless of type). The duration of the timer can be random or deterministic; in either case, it is governed by a timeout parameter.

Note that, with caching enabled, the ability of the system to satisfy an incoming call request depends on the user type, so the different user types do *not* generally experience the same blocking probability. The state of our caching-enabled system will be indicated by a $2N$-tuple of the form $(u_n, c_n)_{n=1}^{N}$, where u_n is the number of active type-n connections and c_n is the number of cached type-n connections, $n = 1, 2, \ldots, N$.

Ideally, we would like to solve the following generic problem:

Caching optimization problem. Let Γ denote the maximum signaling rate that the system can sustain. For fixed values of the arrival rates $\lambda_1, \lambda_2, \ldots \lambda_N$, maximize throughput subject to the constraint that the signaling rate does not exceed Γ.

What are the decision variables in this optimization problem? The discussion of caching schemes from the previous section suggests at least two alternatives. We could decide that, whenever a call completes, we will always cache the connection. In this case, the timeout parameters (which would vary depending on the user type, say) would be the decision variables. Alternatively, we could fix a single timeout parameter for all user types and, whenever a call completes, decide whether to cache or tear down the connection based on user type and system state. This paper concentrates on the second option, although we will briefly return to the first option in Section 5. More complex schemes can certainly be imagined (e.g. allowing timeouts to depend on system state as well as user type in the first approach above, allowing timeouts to depend on user type in the second approach, or allowing timeouts to be adaptive in either approach). However, such complex schemes would be extremely difficult to analyze (even in the single-link case) and, if implemented, might prove equally difficult to tune. In the network context, one would face the additional problem of different "optimal" timeouts for the individual links contained in each route; note also that more complex schemes imply more network overhead, since there is more information to distribute and keep track of. Moreover, we are primarily concerned with a high-arrival-rate, low-blocking regime. In such a regime, state-based caching with fixed timeouts provides a rich space from which to select optimal or near-optimal decisions.

In our effort to maximize throughput, we will naturally seek to achieve a low *average* blocking probability, but this alone gives no guarantee of near-equal blocking probabilities for all user types (although near-equal blocking is clearly a desirable feature). In our generic problem statement, we have not included a constraint that all user types experience equal blocking probabilities; this would add to the difficulty of analyzing the problem and greatly restrict the feasible set. In the simulation results of Section 5, however, we do scrutinize the per-user-type blocking performance of our proposed family of policies.

3. Markov decision process approach

We now turn to the framework of Markov decision processes (MDPs). Our goal is to identify structural features of optimal state-dependent policies, and to design simple, near-optimal heuristic policies using this information. Because of the difficulty of solving constrained MDPs, the explicit signaling rate constraint of the previous section is replaced by a penalty term in the objective function (details will be forthcoming when we discuss the reward structure of our MDP). In our MDP model, the timeouts are independent exponentially distributed random variables, governed by a single mean that does not depend on user type.

In Section 1.3, we argued the merits of basing each cache vs release decision on the current system state. As the capacity of the link increases, the state space's cardinality grows rapidly and the space of policies becomes rich. The MDP framework offers efficient iterative techniques, such as policy iteration, to determine an optimal policy. We have implemented policy iteration numerically and have used this implementation to study the form of optimal policies for $N = 2$ user types and small values of the capacity C (on the order of 10–15).

Before discussing our numerical results, we set out the particulars of our formulation.

3.1. Markov decision process formulation

Our model is a priori in continuous time. Following Lippmann [7] and other authors, we employ uniformization to transform our continuous time Markov chain, and its associated reward structure, to an equivalent discrete time form. We now describe the resulting discrete time MDP. We are interested in the long-term average reward optimality criterion (also called the *ergodic* criterion). We do not give details of the MDP machinery here; Chapter 11 of Puterman's book [9] is a good reference. Our notation is adapted from Puterman's. More detail on the specific model presented here appears in [12].

To completely characterize our MDP, we need to describe the model parameters, the state space, the family of policies under consideration, the event and transition structure, and the reward structure. For convenience, we will give the detailed description for $N = 2$ user types; for $N > 2$ user types, the notation will remain consistent, so no confusion should arise.

Parameters of the model. As before, λ_i will denote the arrival rate of type-i users, $i = 1, 2$, and the call holding time will be exponential with mean 1 regardless of user type. Timeouts for cached connections will be assigned an exponential distribution with mean τ (in general, τ could vary with user type; in this study, we assume all cached connections share the same timeout).

State space. The state space for the capacity-C system, $\mathscr{S} = \mathscr{S}_C$, is the set of four-tuples (u_1, c_1, u_2, c_2) of non-negative integers for which $u_1 + c_1 + u_2 + c_2 \leqslant C$. The state variables u_i and c_i represent the number of active and cached type-i connections, respectively, $i = 1, 2$. Occasionally, we will use the notation $u_i(s) :=$ the number of active type-i connections in state $s \in \mathscr{S}$, and analogously for $c_i(s)$, $i = 1, 2$.

Policy space. In our model, the decision whether to cache a connection upon call completion is the *only* type of decision we allow to vary from one policy to another.

Formally, a policy d is a map $\mathscr{S} \to \{0, 1\} \times \{0, 1\}$. Suppose the current state is $s \in \mathscr{S}$. Writing $d(s) = (a_1, a_2)$, when a type-1 call completes, we cache the connection if $a_1 = 1$ and tear it down if $a_1 = 0$. The second coordinate a_2 is analogously a Boolean value indicating whether we will cache the connection when a type-2 call completes service. If the next event is not a type-i service completion, the value of a_i has no effect, $i = 1, 2$. Note that a_i has no meaning when $u_i(s)$, the number of active type-i connections in the current state, is 0. Our convention will be to force $a_i = 1$ whenever $u_i(s) = 0$, $i = 1, 2$.

A word on terminology is in order: we will not distinguish between the decision rule d and the policy "d^∞" which follows decision rule d at each decision epoch. It is well known (see [9], for example) that there is always a stationary optimal policy (i.e. a policy d^∞ for some d of the form introduced in the last paragraph) whenever state and action spaces are finite. Note in particular that we only have to consider deterministic policies.

Events and transitions. Our MDP features the following events, further distinguished by the associated user type: arrivals, service completions and cache timeouts. The possible state transitions are determined by the following rules:

- Service completion events are the only epochs where we allow ourselves a choice; all of the other branches in the MDP "flowchart" depend only on the state of the system. Of course, upon service completion, the choices are "cache the connection" and "tear down the connection".
- Connection setups are only performed in response to explicit user requests, and only when no cached connection of the appropriate (user) type is available.
- Calls are never blocked when an idle channel is available (a channel is idle if it is neither in active use nor cached).

Reward structure. A reward ρ_i is received each time a type-i call is accepted, $i = 1, 2$. Throughout this study we will have $\rho_1 = \rho_2 = \rho$. A cost c is paid out each time a call setup is performed. Therefore, in state s, the reward for a type-i arrival equals

$$\begin{cases} \rho & \text{if } c_i(s) > 0; \\ \rho - c & \text{if } c_i(s) = 0 \quad \text{and} \\ & \quad u_1(s) + c_1(s) + u_2(s) + c_2(s) < C; \\ 0 & \text{if } c_i(s) = 0 \quad \text{and} \\ & \quad u_1(s) + c_1(s) + u_2(s) + c_2(s) = C. \end{cases} \tag{1}$$

The presence of the cost c encourages caching: as often as possible, we want arriving type-i connection requests to "see" a system with state variable $c_i > 0$, and thereby reap full reward ρ (as opposed to $\rho - c$ or, in the worst case, 0). Of course, caching on behalf of user i increases blocking for the other user (and blocked arrivals garner 0 reward).

Let us aggregate the rewards across states, looking at the ρ and c terms separately. The "revenue" ρ is received whenever a call request is accepted; the penalty c is deducted whenever a call setup procedure is performed. Therefore, for the ergodic criterion (i.e. the long-term average reward), the objective function translates to

$$\rho * \text{throughput} - c * (\text{call setup rate}). \tag{2}$$

The second term is a "Lagrangian" penalty term; this is inserted in the objective function in lieu of an explicit constraint on signaling rate. Note that, had we chosen to penalize call teardowns separately (as detailed above, we "charge" only for setups), the model would not have been enriched. To see this, observe that the call setup and teardown rates are the same because

$$0 \leqslant \text{total \# of setups} - \text{total \# of teardowns} \leqslant C$$

at all times along all sample paths. Thus, assessing a per-teardown cost would merely amount to adjusting the constant c in Eq. (2).

3.2. Numerical evaluation of the discrete Markov decision process and observations on optimal policies

Intuitively, we expect that there will always be an optimal policy d^* that is a *threshold* policy: whenever we have two states s and $\hat{s}$ and a user type i such that

- if the system is in state s and the next event is a type-i service completion, policy d^* caches the associated connection;
- states s and $\hat{s}$ are identical everywhere except in the c_i coordinate;
- $c_i(s) > c_i(\hat{s})$;

then d^* also caches whenever the system is in state $\hat{s}$ and the next event is a type-i service completion. If d^* is a threshold policy, it follows that there is a value $c_i^* = c_i^*(s)$ such that, if the next event is a type-i service completion, d^* caches if and only if $c_i(s)$ is less than or equal to the threshold value $c_i^*(s)$. The threshold value may depend on the type, i, of the departing user and on the other state variables (that is, the state variables other than $c_i(s)$ itself).

Suppose we have an optimal threshold policy. Now we fix the user type i and ask: "does the threshold $c_i^*(s)$ depend on the current state s in a simple way?" In particular, is it sensitive to the number of connections that are currently cached for the *other* user type (e.g. does a given user type's threshold increase with the other user type's cache size?) This question leads us to identify some simple classes of threshold policies as follows.

We define an *active* + *cached* policy to be a threshold policy in which the threshold for user type i is a function of

$$\sum_{n=1}^{N} u_n(s) + \sum_{n=1, n \neq i}^{N} c_n(s)$$

for $i = 1, 2, \ldots, N$. (As before, s denotes the current state of the system.) Further, we define an *active-only* policy to be a threshold policy in which the threshold for user type i depends only on

$$\sum_{n=1}^{N} u_n(s)$$

for $i = 1, 2, \ldots, N$. Lastly, we say that a threshold policy is a *constant threshold caching policy* (CTCP) if, for $i = 1, 2, \ldots, N$, the threshold for user type i is independent of s (note that this definition still allows the threshold to depend on the user type). Note that we give these definitions for arbitrary N, although we have restricted ourselves to $N = 2$ user types in this section for ease of exposition.

We implemented policy iteration numerically using Octave [2]. All of the optimal policies were of the threshold type as expected. We applied our policy iteration code to numerous examples in which there were $N = 2$ user types. Most of these runs featured link capacity $C = 10$ or $C = 12$. Sifting through the Octave output in various ways, we found that the optimal threshold c_i^* is approximately linear in $u_1 + u_2$. We state this carefully in the following empirical observation.

Observation 1. *For $N = 2$ user types and for $i = 1, 2$, the optimal threshold c_i^* is near-constant for fixed values of $u_1 + u_2$. More precisely,*

- *Given a constant U, we often find that $c_i^*(\cdot)$ is constant on $\mathcal{S}_U$, the "level set" of states s for which $u_i(s) > 0$ and $u_1(s) + u_2(s) = U$.*
- *The threshold c_i^* never varies by more than 1 on any level set $\mathcal{S}_U$. That is, $|c_i^*(s) - c_i^*(t)| \leqslant 1$ whenever the states s and t share the same value of $u_1 + u_2$ and $u_i(s), u_i(t) > 0$.*
- *The threshold c_i^* is constant when viewed as a function of the* other *user type's cache size, holding u_1 and u_2 fixed.*

Thus the active-only class of policies (i.e. the set of policies for which the threshold $\hat{c}_i$ is a function of $u_1 + u_2$) contains near-optimal elements; moreover, the subcollection of policies for which the threshold $\hat{c}_i$ is a decreasing linear function of $u_1 + u_2$ also contains near-optimal elements.

Fig. 4 illustrates the content of Observation 1 for a specific example; we emphasize that this example typifies the pattern seen in our suite of runs. We have (arbitrarily) chosen to plot the optimal caching behavior for user type 1. On the set of states for which $u_1 + u_2 = 5$, the threshold c_1^* varies but is always 3 or 4. This is the meaning of the half-shaded dot at coordinates (5,4) on the diagram; compare this to $u_1 + u_2 = 4$, where the diagram indicates a constant threshold of 4. We have marked a threshold of -1 at $u_1 + u_2 = 12$ (the link capacity) to indicate that outgoing connections are not cached here. This is in contrast with the threshold of 0 at the $u_1 + u_2$ value 11, which indicates that an outgoing connection should be cached only if $c_1 = 0$. As a final point of clarification, the absence of a dot at any coordinates

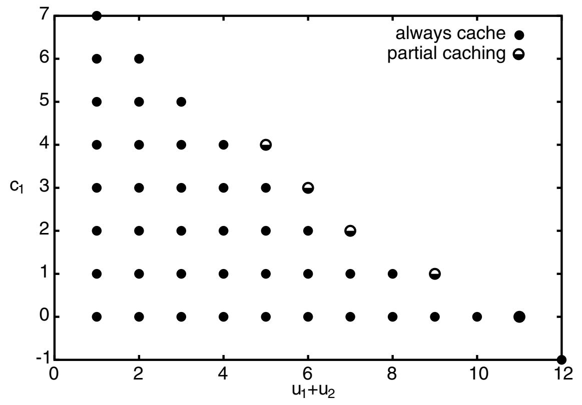

Fig. 4. Plot of optimal policy for user 1 with parameters as follows: link capacity $C = 12$, $N = 2$ user types with arrival rates 5 and 4, per-carried-call revenue 1.0 and per-setup cost 0.03.

means "do not cache when the system is in this (family of) state(s)".

Why should an optimal threshold policy have the features described in Observation 1? First, let us try to understand why caching decisions for each user type should be independent of current cache size for the other user type. When the cache size for one user type is large, why isn't there added incentive to cache connections for the other user type? The paradigm for our optimization model is that the system controls the caching policy in a manner that is transparent to the users; we are not modeling a competitive framework, in which each user type is willing to raise the other type's blocking probability in order to secure its own access to resources. The throughput term in our objective function encourages us to limit caching when it threatens high blocking probability for any of the user types' input streams. Lastly, when the cache size for a given user type is already substantial, adding another connection to the cache intuitively gives little marginal reduction in signaling rate (so, loosely speaking, states with "large" cache sizes will not generally be recurrent states in optimal policies).

The fact that optimal thresholds decrease as a function of the total number of active connections also makes sense. When there are relatively few active connections, sizeable caches do not "cost" much (in terms of increased blocking probability).

When there are many active connections, however, throughput will be compromised unless the non-active channels are, in large part, available to all user types.

3.2.1. Linear threshold caching policy and implementation aspects

So far, we have analyzed the single-link case where multiple connection types with Poisson arrivals of possibly different arrival rates and the same bandwidth requirement share a link with capacity C. The objective was to determine a caching policy that reduces the overall signaling rate without compromising system performance, i.e. blocking probability. Moreover we note that it is desirable that such a policy be fair to the various connection types, i.e. balance the perceived blocking probability and signaling rate. The MDP solution suggests that a state-dependent threshold on the number of cached connections is close to optimal in the single-link case.

In light of the above, we propose the following linear threshold caching policy (LTCP) for the single-link case. A link keeps the following state information: its capacity C, the number of connection types N, the number of ongoing connections for each type u_n, $n = 1, \dots N$ and the number of cached connections of each type c_n, $n = 1, \dots N$. For each type n, there is a state-dependent threshold that is linear in the total number of ongoing connections, $T_n = a_n - b_n \sum_{i=1}^{N} u_i$. Given a departure of a connection of type n, if the current number of cached connections c_n exceeds T_n, i.e. $c_n \geqslant T_n$, then the connection is not cached; otherwise, it is cached. Note that our LTCP is a special case of an active-only policy as defined in the previous section.

Although the parameters (a_n, b_n), $n = 1, \dots N$ might be determined based on approximating the optimal policy given by the MDP analysis discussed above, in practice we propose the following approach to setting these parameters.

First we assume all connection types will share the same parameters (a, b). We will show via simulation in the sequel that this does not compromise our two objectives—performance and fairness. Intuitively, when we apply the same threshold policy to connection types having different arrival rates, one might expect a bias against types with higher arrival rates. However, such connection types are likely to grab more resources and thus equalize such "unfairness".

Second, we choose $b = a/C$. The intuition here is that when the link capacity is full of ongoing connections, no caching should be allowed. The other parameter a is set within the range from $1.25C/N$ to $1.5C/N$, which offers good trade-offs between blocking probability and signaling rate in all the simulations we have run. Details will be discussed in the simulation section.

4. Network model

In the single-link case, we can solve for the exact optimal policy. However in a network, when applying such a caching policy, the optimal policy is hard to obtain. To extend our single link caching policy to a network, we propose a natural extension. The idea is to consider each link on a connection's route independently. Upon connection departure, each link decides whether it should cache the connection or not based on its own state information, i.e. makes a decision based on a single link caching policy. If all links along this route agree to cache the connection, the connection is cached; otherwise, it is not cached.

Although the idea is simple, there are several issues that need to be addressed in the network case.

First, how do we coordinate end-to-end decisions? Although there are different ways to implement an end-to-end decision, a naive design may actually increase the signaling load among links and degrade the performance. Because the goal of the caching policy is to trade bandwidth utilization for lower signaling rate, any factor that will potentially increase the signaling should be addressed with care.

Second, how do we configure the caching policy on each link, given that we may not know the exact number of connection types (i.e., routes, and bandwidth requirements, etc.) going through the link? We need to somehow estimate those parameters that were given in our single-link analysis. How robust is the policy in the presence of estimation errors?

Third, will interactions between a dynamic caching policy and a given routing policy lead to poor performance? For example, it is well known that alternative routing may result in meta-stable states [6] that degrade the system throughput. Similar phenomena may be alleviated by our caching policy alone or in combination with a given routing policy.

Next we briefly discuss how such a policy would be implemented, and might impact performance. The proposed caching policy has the following characteristics:

- Caching decisions are made upon connection departures.
- If a connection is cached, all the resources along its route will be held.
- Only connections on shortest path routes in terms of hop-count will be candidates for caching, i.e., connections following alternative, longer routes will not be cached.
- A connection is cached if and only if each link along its route agrees that it should be cached. If any of the links fails to do so, the whole route will be torn down without caching.
- Each link makes its own decision whether to cache a connection or not based on the LTCP policy.

This policy addresses the above noted problems associated with implementing network-level caching. We observe that there are two ways to obtain the necessary information for the caching policy: by coordinating caching decisions via signaling, or by using possibly outdated link state information at the source. If we use the first solution then we save the setup cost, but implementing the caching policy may have a signaling overhead almost equivalent to that of tear down. Therefore we believe that the second approach is preferable for implementation as long as network loads are sufficiently stationary. However, we note that for connections managed via the second approach, the source needs to acquire caching thresholds for links along the route upon connection setup (by "piggybacking" additional information on signaling messages that acknowledge the completion of a setup, which would increase the size of certain messages but would not require additional signaling exchanges).

To configure the caching policy on a given link, we will only consider caching connections routed on shortest path routes. The rationale for doing this is as follows: the implied cost associated with cached connections on non-shortest path routes may not warrant caching. Alternative routed connections tend to consume more resources. In a system with reasonably heavy traffic on every link, the implied cost, which is associated with the blocking probabilities of all other traffic sharing bandwidth along this route, could be much higher than the benefit of caching such a connection. Another reason is because our caching policy is state dependent. In a network with a complex topology, keeping the information of all possible routes can be cumbersome and the approach would not scale. Restricting caching to connections on shortest path routes reduces the amount of state information that needs to be stored.

In a large network, it may be hard in practice to determine whether a connection is routed on a shortest path route. Here we assume that all the shortest path routes for given source/destination pairs are well known to sources, i.e., can be precomputed based on the network topology.

Finally, we claim that favoring the shortest path routes in our caching policy will have the similar effect to that of "trunk reservation", which can enhance the throughput. By reserving more resources, i.e. caching connections routed on the shortest path routes, there is less of a chance that the system will fall into the meta-stable states where alternative routed connections are dominant and connections are experiencing high blocking probability. This claim matches the observations in our simulations.

Scalability. One of the main motivations for this work is to find ways of reducing signaling load *within* a softswitch. That is, we wish to ease the burden of setting up and tearing down a connection between ingress and egress nodes (we will call these *media gateways*) each time a call request arrives from the outside world. In order to be useful, our scheme must be scalable. The main issue here is the size of the state space associated with each link: we need to make sure that this size is manageable for the intended application, or in other words to make sure that the number of connection types that might be cached on any given link is not exhorbitant. We will regard "connection type" as synonymous with "O–D pair" in the following sense: if two shortest path connections for the *same* O–D pair share a common link, that link will not distinguish between the two connections but will instead count them as being of the same type for the purpose of caching decisions. This is reasonable; in particular, it does not abridge any link's "right" not to cache a connection based on the amount of resources it has allocated to the associated O–D pair. Now the total number of connection types that traverse our softswitch is clearly an upper bound for the number that might be cached on any given link. The former quantity is quadratic in n, the number of media gateways (it is $\binom{n}{2}$). It is important to note that the number of connection types does *not* depend on the number of switching nodes residing in the "interior" of the softswitch fabric (i.e. nodes that are not ingress/egress points). The number of media gateways is rarely large; in fact 10 is a substantial number in today's softswitch deployments. This is because

(1) For a number of years to come, softswitches will be deployed as islands in a circuit-switched world (and will therefore be required to conduct signaling interactions with the "circuit-switched world" on a per-call basis).
(2) Control of a large number of media gateways presents scalability problems of its own. (This is particularly applicable to large telephone service providers, who want each media gateway to serve a large number of end users, and for whom reliability is absolutely paramount. A small, specialized service provider might want to deploy a softswitch featuring media gateways that are smaller but much more numerous; our scheme is less suitable for the latter scenario.)

In the softswitch telephony context discussed above, we have assumed that there is only one category of traffic (i.e. voice). So the total number of connection types is the same as the number of O–D pairs. If this assumption is removed (e.g. our softswitch supports video sessions as well as voice calls), the number of connection types scales linearly with the number of traffic categories: no link will carry more than $\binom{n}{2}c\binom{n}{2}$ connection types, where c is the number of traffic categories in a heterogeneous network. Moreover, the upper bound given here does not tend to be tight: we would *not* expect every connection type to appear on every link. Indeed, we never cache a connection when it is not a shortest path. As a concrete example, assume that a large softswitch matches the NSF T1 topology given in Fig. 10. The MI-UT link is traversed by shortest paths for 23 of the $\binom{16}{2} = 120$ O–D pairs; this is the maximum of any link.

Suppose voice calls or video sessions traverse multiple softswitches. This scenario takes place when forced by the scalability limitations noted in items 1 and 2 above, or when softswitches owned by two different service providers are interconnected. In this case, the softswitches would independently implement separate, non-interacting caching policies: although a softswitch might choose to cache the segment of a multi-switch bearer path that traverses its fabric (e.g. in the form of an MPLS Label Switched Path), segments traversing multiple softswitch fabrics would never be cached.

5. Simulation

In order to evaluate the performance of the caching policy developed above, we ran two sets of simulations. Throughout this section, timeouts are deterministic; whenever more than one cached connection is available to serve an incoming call request, we choose the "oldest" connection, i.e. the connection that is closest to timing out.

The first set of simulations corresponds to the single-link case with various parameter setups. The simulations are intended to illustrate that the linear threshold caching policy (LTCP) class is better than other classes of policies, such as the constant threshold caching policy (CTCP) class and the active + cached class (these were defined in Section 3.2); we present graphs summarizing our results. The superiority of LTCP to the timeout-only approach (in which outgoing connections are always cached—this was introduced in Section 1.3) was also confirmed by our simulations. In the interest of space we do not present details of our timeout-only simulations beyond briefly describing our setup as follows: all of the user types had identical arrival rates, a symmetric situation in which there was no apparent reason for timeout parameters to vary with user type. Thus each simulation was configured with a single timeout parameter; in a series of simulation runs, we searched empirically for an optimal setting of this parameter. No parameter value, however, yielded timeout-only performance commensurate with the performance of a well-tuned LTCP.

The second set of simulations corresponds to the network scenario. Our aim is to show that the proposed generic caching policy can significantly reduce the signaling rate without degrading network utilization and validate the proposed design/configuration heuristics for the network case. In the network case, we will consider both static and alternative routing schemes. In view of the scale of soft-switch network, we only have a simple triangle topology in this paper to show our key observations.

Performance was evaluated in terms of the trade-off between signaling rate and blocking probability. For example, in Fig. 5, the x-axis is the signaling rate and the y-axis is the blocking probability. Each curve corresponds to the performance achieved by a given policy as its operational parameters change accordingly. Each point on the curve represents the (signaling rate, blocking probability) given a particular threshold parameter; for example, CTCP(2) means that the constant threshold of CTCP is 2 and LTCP(1.0) means that the parameter a of LTCP, which determines the slope of the linear threshold as discussed in Section 3.2.1, is 1.0.

5.1. Single-link case with identical connection types

Although as mentioned in Section 1.1 we assume that different connection types have the same resource requirements, they may have different arrival rates—in this case we call the simulation

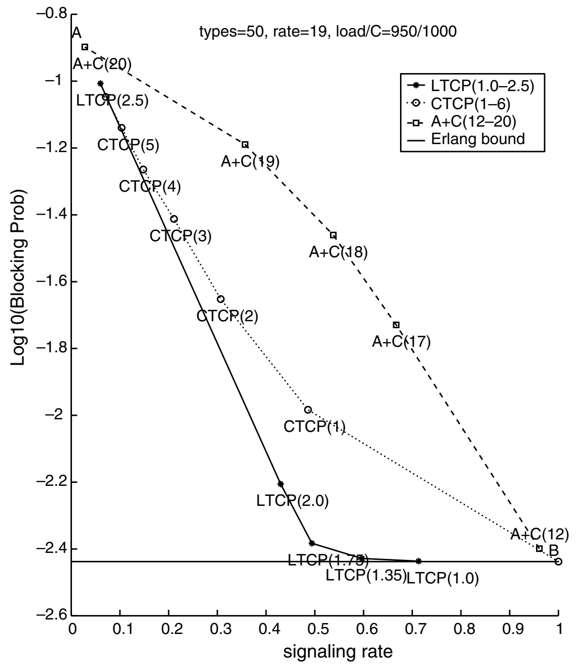

Fig. 5. Single-link case with connection types having homogeneous loads.

setup heterogeneous; otherwise we say it is homogeneous.

5.1.1. Traffic types with homogeneous loads

We consider the following setup: the link capacity C is 1000 units. There are 50 types; each has a Poisson arrival with offered load 19 Erlangs and a bandwidth requirement of 1 unit per connection. Therefore the average load on this link is $50 \times 19 = 950 < C$ so that the link is not overloaded. Four possible classes of caching policies are considered: no caching, LTCP, CTCP and active+cached. No caching corresponds to a point with 100% signaling rate on the graph. We derived a horizontal line from this point to indicate the theoretical lower bound on blocking probability of the single-link case given the offered load and the link capacity.

As seen in Fig. 5, LTCP has the best performance among all classes of policies. Given the same blocking probability, LTCP always has the minimum signaling rate. Therefore among all the policies, this policy is the closest to optimal. For example, in the region where the blocking probability is acceptable (<0.01), the signaling rate is reduced by more than 60%. All the policies behave similarly at the two extremes (A and B in Fig. 5). These are not of interest to us because one extreme has high signaling rate close to 100% due to insufficient caching and the other has high blocking probability due to excessive caching. Since the active + cached policy has poor performance, we will ignore it in later sections and focus on comparing our LTCP with CTCP.

Another advantage of LTCP is its robustness. Its performance is relatively stable in a large range of values for the parameter *a*, e.g. 20–40. This helps in selecting a "good" value for the parameter *a* in more complex cases. By contrast, CTCP is highly sensitive to the parameter setting. For example, by comparing the points CTCP(2) and CTCP(3) in Fig. 5, we can see that CTCP has 80% higher blocking probability when the constant threshold is changed from 2 to 3. As shown in both this simulation and later ones, typically a good value for *a* in LTCP is between $1.25C/N$ and $1.5C/N$. This heuristic was empirically validated in all the cases that we considered.

5.1.2. Multiple traffic types with heterogeneous loads

Next we simulated a link shared by multiple traffic types with heterogeneous loads as follows:

The link capacity C is 1000 units. There are a total of 30 traffic types with different arrival rates. Group H has 10 types with offered load 60 Erlangs; group M has 10 types with offered load 25 Erlangs and group L has 10 types with offered load 10 Erlangs. The total arrival rate on this link is therefore $60 \times 10 + 25 \times 10 + 10 \times 10 = 950 < C$.

In this simulation, we measured both the blocking probability and signaling rate for each group as well as the overall averages. As mentioned in the previous section, we applied the same threshold function for all traffic types. For comparison purposes, the performance of traffic types with uniform loads and the same total offered load of 950 Erlangs is also shown in Fig. 7.

As seen in Fig. 6, there is a degree of unfairness among different groups, but it is not very significant. Traffic types with higher arrival rates experience higher blocking probability and signaling rate. The amount of resources they grab cannot catch up with their higher arrival rates, i.e. higher bandwidth requirements. We attempted to eliminate this unfairness by fine-tuning the parameter *a* for different groups. However, if we modified the policy only via the parameter *a*, fairness proved difficult to achieve. Based on these and other results, we concluded that a reasonably good solution to achieve approximate fairness would be to set the same value of *a* for all types. The estimation of $a = 1.5C/N$ leads to good performance even if traffic loads are heterogeneous. For this simulation, $a = 1.5 \times (1000/30) \simeq 50$. In both Figs. 6 and 7, around the region $a = 50$, the signaling rate is reduced remarkably by 60%, and the blocking probability remains almost the same as the non-caching lower-bound. Unfairness among the three groups is negligible in this operating regime.

Again, comparing the average performance for the simulation results shown in Fig. 7, we see that LTCP outperforms CTCP, especially in the low-blocking regime of interest. (We comment that the

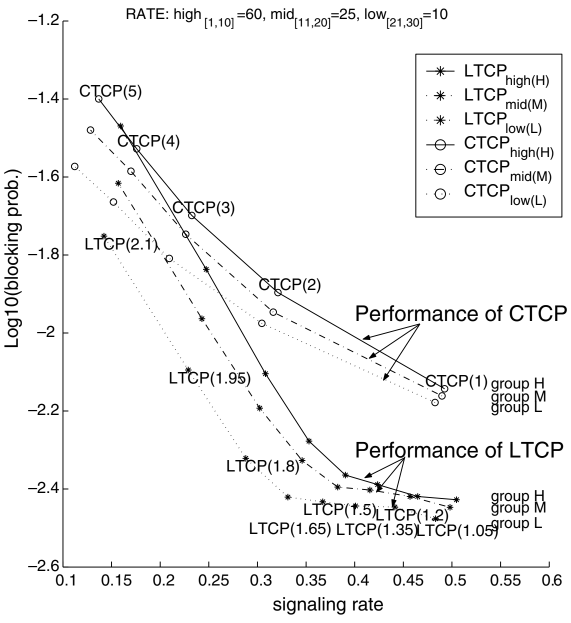

Fig. 6. Single-link case with types having heterogeneous loads.

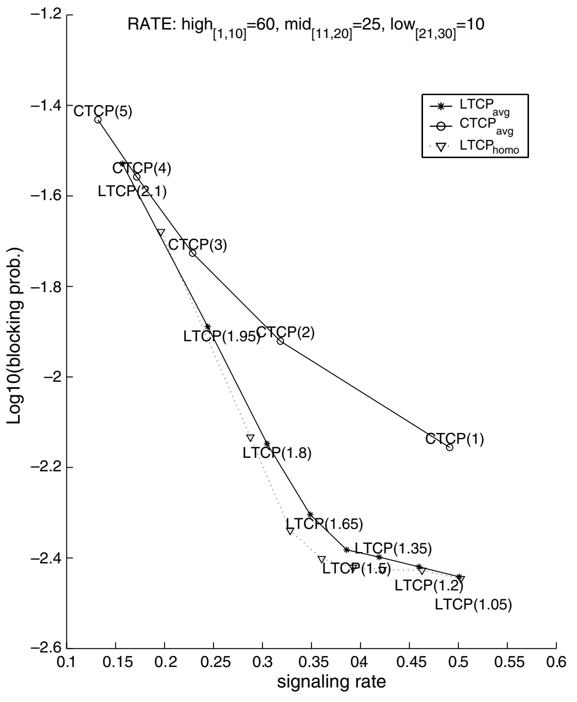

Fig. 7. Single-link case with types having heterogeneous loads (average performance).

homogeneous case in Fig. 7 has 30 user types; this is a different curve than in Fig. 5, where there are 50 user types. For both curves the total load is 950 Erlangs.) Moreover, LTCP has similar performance in both the case of heterogeneous traffic and the homogeneous traffic types with the same overall arrival rate. Therefore, we conclude that LTCP can deal with heterogeneity without degrading performance or losing fairness.

5.2. Network case

5.2.1. Simple topology with fully connected 3-node network

Our first network topology is shown in Fig. 8. Topologies such as this, in which a small number of switching elements are fully meshed, are not uncommon in softswitch deployments. In the ATM case, the nodes pictured could be performing AAL2-layer switching, inhabiting a larger network in which the other nodes (not shown and indeed not of interest here) function only at the ATM layer. We set up the simulation as follows:

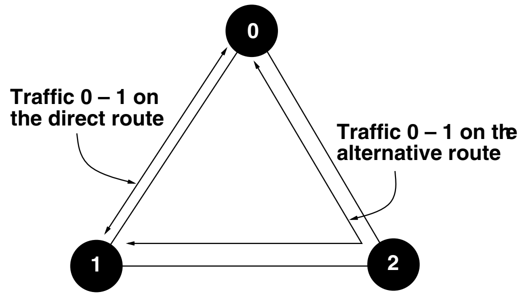
0
1
2
Traffic 0 – 1 on the direct route
Traffic 0 – 1 on the alternative route

Fig. 8. Fully connected 3-node network.

Each link has a capacity 1000. Between each two nodes, there are 10 traffic types with the same Poisson arrival and offered load 90 Erlangs. Therefore, without considering alternative routing, the average load on each link is $10 \times 90 = 900 < 1000$. If any blocking occurs on the direct path, the connection will try the 2-hop alternative route. Blocked calls are counted for connections that fail on both the direct route and the alternative route. As stated in the proposed caching policy, alternative routed connections are not cached.

For each link, the LTCP initializes its parameters a given the number of connection types on shortest path routes. For comparison, we implement CTCP such that all links keep the same fixed constant threshold on the maximum number of cached connections.

The simulation results are shown in Fig. 9. The horizontal line is the blocking probability when no caching is implemented. As seen in Fig. 9, LTCP again yields better performance than CTCP. Also the proposed heuristic for setting the value of parameter a is still good in the network case.

One difference in the network case vs the single-link case is that the blocking probability is not strictly decreasing in the signaling rate as the parameter a for LTCP is varied; the no-caching regime no longer offers a lower bound on blocking probability in the network case as it did for the single-link case. We argue that by caching direct routes, we force some connections to be blocked on their alternative routes while they could get through when caching is not used. For this topology, alternative routes will consume two times the resources as those directly routed. Therefore

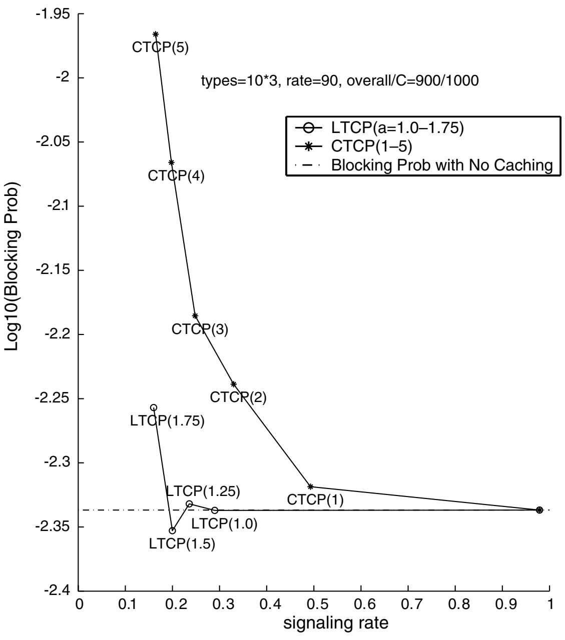

Fig. 9. Performance of caching policies in a fully connected 3-node network.

limiting the admission of alternative routes can reduce the overall blocking probability.

Finally, in our simulation, we did find cases where the network transits from a "stationary" low-blocking state to a high blocking state, i.e. the meta-stable behavior. Since these cases happened only for low or zero cache levels, this suggests that our caching policy does help to avoid such behavior.

5.2.2. Topology based on NSFNET T1 backbone

Finally we compare LTCP and CTCP on the topology of the NSFNET T1 backbone shown in Fig. 10. The simulation is set up as follows.

As before, each link has capacity 1000. Between any two directly connected nodes, there is a "background" traffic type with offered load 330 Erlangs. Also across the network there are six traffic types running through longer routes each with offered load 330 Erlangs.

The caching policy only considers caching connections on shortest path routes in terms of hop-count. We ran simulations for two different scenarios. In the first one, connections are routed on static shortest path routes. A connection attempt fails if it is blocked on its associated route. In the second scenario, if any connection is blocked on its shortest path route, a dynamic routing algorithm is used to find the current "shortest path" where the link metrics are given by 1/residual bandwidth. A connection attempt fails if it is also blocked on the alternative route. Again, these alternative routes are not cached. We are interested in the performance of the caching policy on links that are heavily congested. With the above loads, the link between nodes TX and MD is in fact the most heavily loaded. The following results assess the signaling loads on this link.

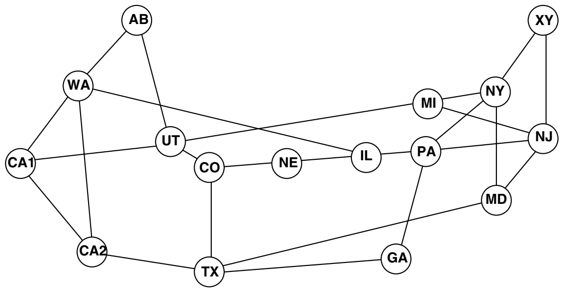

Fig. 10. NSFNET T1 backbone topology.

From Fig. 11, we can make two key observations. First, LTCP outperforms CTCP in reducing signaling rate given the same blocking probability. In Fig. 11, which shows the performance of the two caching policies on the most heavily loaded link, this improvement is marginal. However, in terms of the average performance of the system, not shown in the figure, LTCP has a much better performance. For example, when the average blocking probability equals 0.021, the LTCP yields a signaling rate of 0.024 and CTCP has a much higher signaling rate of 0.197. The reason for this is that when links have different loads, LTCP can dynamically adapt itself to the link state and therefore approximate the optimal cache level better than CTCP, which always keeps a fixed threshold. Limited by the number of types in our simulation, the improvement on reducing the signaling rate shown in Fig. 11 is perhaps optimistic, i.e. when more connection types co-exist on the network, the reduction in signaling rate would be less. However, we still can expect that in a real

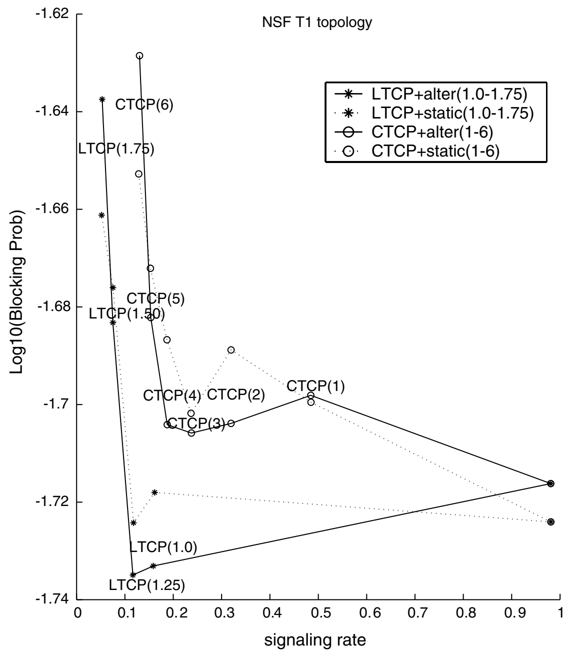

Fig. 11. Performance of caching policies on the NSFNET T1 backbone topology.

network, with a good value for parameter a, i.e. $a \approx 1.25C/N$, LTCP will be very useful in reducing signaling loads without compromising blocking probability.

The second observation is that, when a caching policy is applied along with the alternative routing scheme, the blocking probability can be even smaller than the non-caching case on the most heavily congested links. The reason for this behavior is similar to the case in the previous simulation of 3-node network. Since the number of connections that are on shortest path routes is dominant compared with the number of connections on alternative routes, the caching policy will be biased against those connections on alternative routes by reserving resources for connections on shortest path routes and therefore yield better average performance.

6. Conclusions

In this article, we motivate connection caching and discuss possible schemes for deciding when to cache an "outgoing" connection and when to tear down an unused cached connection. We argue that state-dependent caching is necessary for best performance and propose a linear threshold caching policy (LTCP) that appears to be near-optimal in the single-link case. Through simulations, we verify that LTCP performs well in the network context, and is relatively insensitive to errors in parameter estimation. Thus connection caching with the LTCP heuristic is a promising technique for avoiding signaling congestion in networks that suffer from inadequate signaling capacity.

Acknowledgements

This work is supported by NSF Career Award NCR 96-24230, by SBC Technology Resources, Inc., by Cingular Wireless and by an Intel Technology for Education 2000 equipment grant.

References

[1] G. Apostopoulos, R. Guerin, S. Kamat, S.K. Tripathi, On reducing the processing cost of on-demand QoS path computation, Journal of High Speed Networking 7 (2) (1998) 77–98.
[2] J.W. Eaton et al., GNU Octave. Available from <http://www.che.wisc.edu/octave/>.
[3] J. Wang, A survey of web caching schemes for the internet, ACM Computer Communication Review 29 (5) (1999) 36–46.
[4] G. Barish, K. Obraczke, World Wide Web caching: trends and techniques, IEEE Communications Magazine 38 (5) (2000) 178–184.
[5] C. Aggarwal, J.L. Wolf, P.S. Yu, Caching on the World Wide Web, IEEE Transactions on Knowledge and Data Engineering 11 (1) (1999) 94–107.
[6] F.P. Kelly, Loss networks, Annals of Applied Probability 1 (3) (1991) 319–378.
[7] S. Lippmann, Applying a new device in the optimization of exponential queueing systems, Operations Research 23 (1975) 687–710.
[8] M. Peyravian, Network path caching: issues, algorithms, and a simulation study, Computer Communications 20 (1997) 605–614.
[9] M. Puterman, Markov Decision Processes, Wiley, New York, 1994.
[10] V. Nguyen, On the optimality of trunk reservation in overflow processes, Probability in the Engineering and Informational Sciences 5 (1991) 369–390.

[11] Y. Serbest, S. Park, A. Sang, S.-Q. Li, A simple adaptive SVC caching scheme for voice trunking over ATM (VTOA) applications, in: Proceedings of IEEE INFOCOM 2000.

[12] M. Stafford, Online search and connection caching, Ph.D. Thesis, The University of Texas at Austin, May 2000.

Matthew Stafford is a principal member of technical staff at Cingular Wireless. His current research interests are in data networking, particularly data services in 3rd generation wireless networks. He holds a Ph.D. in Mathematics from Northwestern University and a Ph.D. in Operations Research from the University of Texas at Austin.

Xiangying Yang received his B.S. in Electrical Engineering from Tsinghua University, China in 1998 and M.S. in electrical and computer engineering (ECE) from University of Texas at Austin in 2000. He is now a Ph.D. candidate in ECE department at University of Texas at Austin. His research interests include web caching, peer-to-peer applications and wireless sensor networks. He is a recipient of Texas Telecommunications Engineering Consortium (TxTEC) Fellowship in year 2000, 2002 and a member of TauBetaPai.

Gustavo de Veciana received his B.S., M.S., and Ph.D. in electrical engineering from the University of California at Berkeley in 1987, 1990, and 1993 respectively. In 1993, he joined the Department of Electrical and Computer Engineering at the University of Texas at Austin where he is currently an Associate Professor. His research focuses on issues in the analysis and design of telecommunication networks. Dr. de Veciana has been an editor for the IEEE/ACM Transactions on Networking. He is the recipient of a General Motors Foundation Centennial Fellowship in Electrical Engineering and a 1996 National Science Foundation CAREER Award, and co-recipient of the IEEE Bill McCalla Best ICCAD Paper Award for 2000.