

DEPARTMENT OF ELECTRICAL ENGINEERING TECHNICAL REPORT

THE COLUMBIA HOTSPOT RESCUE SERVICE: A RESEARCH PLAN

Ed Coffman, Predrag Jelenkovic, Jason Nieh and Dan Rubenstein

CU/EE/TR 2002-05-131

Columbia University
Department of Electrical Engineering
500 W. 120th Street, MC 4712
New York, NY 10027

<http://www.ee.columbia.edu>

Copyright © 2002 – The Trustees of Columbia University in the City of New York
All Rights Reserved

The technical reports in this series are considered to be semi-formal.
The ideas expressed are solely those of the authors, and questions about the content
should be directed to them.

The Columbia Hotspot Rescue Service: A Research Plan

Ed Coffman*[‡] Predrag Jelenković* Jason Nieh[†] Dan Rubenstein*[†]
*Department of Electrical Engineering [†]Department of Computer Science
[‡]Department of Industrial Engineering and Operations Research
Columbia University
New York, NY

Technical Report
Computer Networking Research Center
Columbia University
February, 2001

1 Introduction

Internet performance is unpredictable, and users often find the level of service inadequate. We expect this problem to worsen as the Internet continues to grow, both in the number of users and in the stringency of the requirements of new applications. Intolerable levels of service stem from a lack of sufficient resources being targeted to locations where service quality is lacking. In a system as complex as the Internet, the insufficient resource can be of many types. It may be a server with insufficient processing power to handle a large load of requests. It may be a router at the edge of the network that is forced to carry a heavy load to or from its local area network. It may be a router in the middle of the network that must carry a large load, perhaps connecting a set of ISPs, such that upgrading its performance does not directly increase any company's profits. Lastly, it may be an attacker intentionally trying to bring down a portion of the network. Commonly, the periods of insufficiency are short-lived. In other words, the resources allocated within a network to support a service are sufficient most of the time, but every so often and quite suddenly, there may be a sharp increase in demand, and the resource quickly becomes overwhelmed. Such an event is called a network *hotspot* [1, 2].

There are several remedies currently available to deal with network hotspots. The easiest, most common solution is to make do with the network as is and deal ad hoc with the unpredictable level of service. A second remedy that works in theory is to “build a better network” in which hot spots do not occur. Here, each network and server component must be provisioned with enough resources to handle the maximum possible short-term overload. However, such overprovisioning is extremely costly, making it an infeasible solution for the majority of network servers and users. Another remedy is for companies to offer content delivery services, such as caching or access to additional network bandwidth [3]. The content delivery company deploys servers throughout the network, and then profits when those wishing to provide information efficiently are forced to pay for access to these caches and for bandwidth, basically putting the expenses of the information provider at the mercy of the content delivery companies. This leads to the situation that we have in today's Internet, in which large companies with tremendous financial resources buy efficient delivery, and where smaller companies, academic institutions, non-profit organizations, and individuals wishing to deliver content but who cannot afford the content delivery companies' high premiums are left in a position vulnerable to serious lapses in their ability to reach clients.

Ironically, the resources necessary to provide support to all of these entities, including the large, wealthy companies, are already present within the Internet. This is even the case if one excludes the resources of the

content delivery companies. While hotspots cause great displeasure to a large portion of network users and seem to pose a continuing threat on a global scale, the fact is that, at any given time, the resources whose overloading causes hotspots are a tiny piece of the network. At the same time, under-utilized resources that could be used to alleviate the hotspots are densely scattered throughout the rest of the network, under the control of a seemingly unbounded number of organizations. These resources are presently unable to alleviate hotspot conditions simply because there is a lack of a coordinating mechanism that can direct the under-utilized resources to the aid of the over-utilized ones.

What is needed is a system that can quickly identify hotspot buildup in the network, locate a set of under-utilized resources, and find a way to improve service at the hotspot location by shifting some of the service burden to under-utilized resources. We refer to the service that offers on-line identification of and relief from hotspots as a *Hotspot Rescue Service (HRS)*. The goal is to study and eventually to build a fast and efficient HRS that does not require a third party to provide additional resources. At a high level, we envision two approaches to designing such a service, neither requiring additional hardware. A *Server-based* approach requires additional resources at strategic times from those servers that wish to utilize the HRS to ensure that their objects reach all customers in an efficient manner. These servers provide spare processing and buffering resources (when available) to the HRS that can be used by the HRS to detect and possibly alleviate hotspots throughout the network. A *Peer-to-Peer (P2P)* approach requires resources at strategic times from those clients that wish to utilize the HRS to ensure that they can efficiently receive all objects. These clients must provide processing and buffering resources that can be used by the HRS to detect and alleviate the hotspots. We discuss the details of the server-based and P2P approaches separately for clarity of presentation. However, we note that an important goal is to design the HRS so that both approaches can be combined effectively within one system.

The rescue service has four broad phases of operation:

Phase 1: Prediction / Identification Network hotspots must be predicted or at least identified by the rescue service. The server-based approach accomplishes this by deploying daemons on servers that observe network traffic and server processing conditions, looking for signs of developing hotspots. These daemons, either in isolation or in a distributed fashion, notify the service of a potential hotspot and perhaps send along statistics about the hotspot, or the manner in which resources must be made available in order to alleviate the hotspot. The P2P approach accomplishes this by installing *Hotspot Identification Plugins (HIPs)* within participating clients' browsers. A browser's HIP transmits pages stored within the browser's cache and exchanges information with nearby HIPs, such as cache content. Using distributed algorithms, these HIPs can detect hotspots by identifying items or locations of items whose popularity is rapidly increasing.

Phase 2: Replication After predicting or detecting a hotspot, the rescue service must next locate and assign resources within the network to alleviate the hotspots. A simple server-based rescue system may involve replicating the objects within other network servers at alternative locations, whereas a P2P service would identify those clients whose caches will maintain copies of various hot objects. A more advanced rescue system might also choose to reroute traffic or redistribute the network traffic load, or place reservations for bandwidth within routers.

Phase 3: Notification Various regions of the network may require notification that a rescue service has been deployed. For instance, in the case of replicated services, a server-based system would inform clients of the location of a replicated object. When the hotspot is a result of something other than an overloading of the request queue, a simple redirect is sufficient. Otherwise, a more complex solution involving modifications to the mapping process that associates an IP address with an object (e.g., DNS) may be required. Similarly, a P2P approach would proactively announce to clients the set of objects which are hot, and would indicate alternative locations from which these copies can be retrieved.

Phase 4: Termination When the hotspot has passed, the service must release the resources whose prior use prevented overload conditions within the network. This allows application of these resources to alleviating subsequent hotspot activity. This involves contacting the points where objects are replicated, and informing those points of the objects that may be released from their cache. Furthermore, clients that had previously been informed of the cached locations of objects must have this information updated.

The rescue service must account for the differing network requirements of the various forms of media that are carried through the Internet. We divide the objects that the service is designed to handle into three different classes:

Small Web objects: This object type is a small (several kilobyte) file. For these objects, the HRS must maintain acceptably small *delivery delays*, the times between object requests and corresponding delivery times. For these small files, the time taken to identify the file's storage location contributes significantly to the delivery delay. Hence, a major focus of the HRS is to minimize the time it takes to identify the replicated object. Since the overhead of the actual data transfer is not much larger than the overhead of establishing a connection, we suspect that hotspots for this class of objects will often be the result of an overload of requests for the object at the server or an overload of bandwidth-limited points in the network.

Large data files: Objects such as software packages, software updates, and MP3 music files can be large in size (currently ranging from a few megabytes to several gigabytes.) As above, the goal of the HRS is to keep the object's delivery delay small. However, the transfer time and server processing associated with these files is large in comparison to the time and processing required to establish a connection. We suspect that hotspots for this class of objects will be the result of insufficient resources to handle the transfer of the data at either the server or at bandwidth-limited network points. We expect that the sizes of files that people attempt to transfer will grow proportionally to transmission speeds of the network and servers, such that a large data file class will continue to exist in the future.

Stored streaming media files: Stored streaming media transmissions, such as video or audio broadcasts, have quality of service (QoS) requirements such that reduction of delivery delay is not the measure of greatest interest. Instead, the HRS must ensure that each request results in a connection for which the available network bandwidth and server processing is sufficient during the lifetime of the connection.

These three classes contain objects whose content is static over moderate time-frames of at least tens of seconds. There are other important classes of objects that the initial design of the HRS does not handle, such as dynamically generated web pages or live transmissions that are difficult to replicate. Furthermore, there are issues involving security, such as authentication of replicated objects and the prevention of abuse of the HRS to cause a denial of service (DoS) attack.

Finally, the HRS must be able to find and alleviate different forms of hotspots. In particular, we distinguish between hotspots created by an overburdening of the end-system server and those that place an excessive load on some resource within the network. Alleviating the overburdening of an end-system requires finding a replica with sufficient unused processing capacity. In this case, the location within the network is arbitrary. For alleviating excessive load on a portion of the network, a replica must be identified such that its transmission path to the client does not traverse the overloaded network region.

1.1 Research Plan

Our recommendation is to build a collaborative and self-organizing distributed network rescue service that will detect and potentially eliminate hotspots. The architecture of the system is outlined in Section 2, and incorporates two main modules: hotspot early detection and response, i.e., rescue operation, which are discussed in Sections 3 and 4, respectively. Section 5 summarizes related problems that we expect to eventually tackle. Concluding remarks and the broader impact of this research are presented in Section 6; subsequent pages then give more specific research agendas.

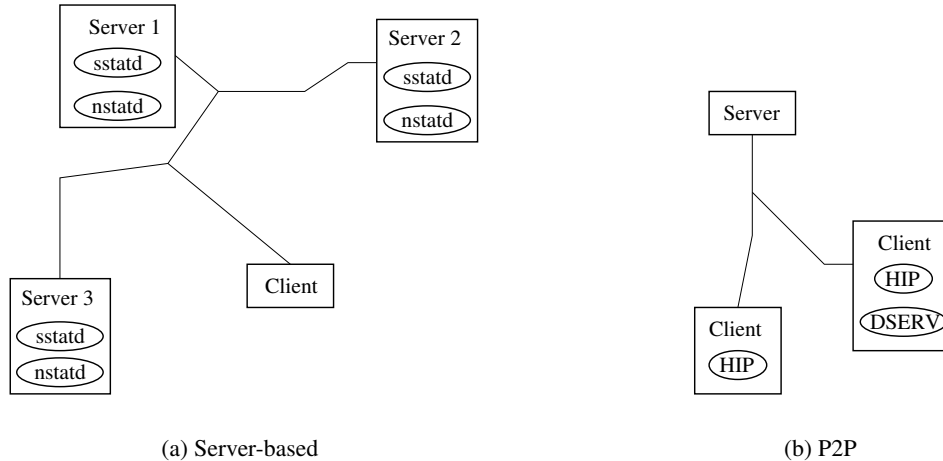


Figure 1: Proposed architectures

2 General Architecture

Hotspots occur when server, link, or switching capacity becomes overloaded, or there is a network device failure. The discussion below focuses on detecting hotspots caused by a rapid relative increase of server and/or network load, which we refer to as a "temperature" increase. Figure 1 visualizes at a high level small instantiations of two possible architectures that can support the HRS. Figure 1(a) illustrates the server-based architecture in which receivers require no additional functionality beyond what is currently required to receive objects within the Internet. Such an architecture allows future servers to protect their transmissions from hotspots, even when clients use unmodified versions of today's standard browsers.

Each server runs two daemons, `sstatd` and `nstatd` which respectively monitor server and network resources and handle replication of objects between servers. A `sstatd` daemon has two monitoring responsibilities, each of which monitors the resource availability of servers (as opposed to of the network). The first is to monitor the server processing characteristics on which it resides. The processing required by the daemon itself can vary. At the low end, the daemon can simply sample server logs to determine increases in accesses for a given object. Additionally, it can periodically sample the current server load, the number of active connections, and these connections' running times. At the high end, it can actively monitor incoming and outgoing packet traffic. Via this monitoring, the daemon attempts proactive detection of hotspots. However, the daemon may fail to detect an oncoming hotspot. Indeed, once the server load has reached a high enough temperature, a daemon with a low scheduling priority may effectively cease operating, making it incapable of detecting hotspots. As a failsafe, the daemon's second responsibility is to monitor the performance of nearby servers that participate in the HRS. This is done via the transmission of periodic pings between servers so that the servers can monitor each other's status!¹ A server must respond to a ping request. Note that the response to the ping can be the transmission of a popular object, allowing the querying server to cache an object of a nearby server before a hotspot occurs. The rapid response to a ping is an indication that the responding server's load is manageable. Delays or failed responses serve as an alert that the pinged server might be in an overloaded state.

The `nstatd` daemons monitor the status of the network between the the server on which it resides and recently contacted servers and clients. This daemon is responsible for identifying congestion inside the network (i.e., at a network router). The daemon uses information from probes to other servers as well

¹Note that our usage of the term "ping" means an attempt to establish contact. The ping need not be an `ICMP ECHO REQUEST` as would be generated by the Unix `ping` command.

as status information obtained by observing communication quality with geographically distributed clients to develop a map identifying congested areas of the network. In principle, such information could also be provided by a separate service, such as the IDMaps [4, 5, 6] or SPAND projects [7]. In addition, one can apply techniques that identify bottleneck bandwidth capacities [8, 9, 10, 11], or techniques that identify common congestion points between pairs of servers to various network points [12, 13]. The `nstatd` daemons exchange recent congestion information with one another and use this information to determine where content needs to be replicated so that it can be accessed efficiently by clients throughout the network.

Figure 1(b) illustrates the P2P architecture. The approach builds on the P2P networking method that has been very successful in Napster for distributing digital music. The key premise of this technique in the current setting is that hotspots develop as a result of many clients attempting to access the same content. As more and more clients access the given content, the web server becomes less and less able to service the requests. However, as more and more clients access the given content, the content increasingly (and automatically) replicates on client machines used for accessing the web content. As a result, if the clients' machines can be used for providing the hot content and if requests for the given content can be redirected to the client machines, the hotspot at the server can be eliminated. In this way, hotspot response becomes self-organizing and self-regulatory.

Each client installs a HIP plugin within its browser. The HIP plugin caches those objects that are accessed by the hosting client, and transmits these objects when requested to do so by another client (i.e., the client behaves like a server and transfers the object). In addition, these HIP plugins exchange information regarding the objects that are cached locally. High temperature objects will naturally be distributed among the various client caches, and server load will be reduced as a result of this distribution among the various clients.

In addition, participants in the rescue service can also install a *Directory Service (DSERV)* that keeps track of those client machines that have downloaded the given web objects and determines which of those client machines to send the redirected content request. This redirection decision is made to provide good load balancing across client machines, taking into account the available client and network resources for each client machine that has cached the content.

The strength of the server-based architecture is that it allows servers to protect the delivery of their content without imposing an additional caching and transmission burden on clients. This may prove especially useful when dealing with backward compatibility, wireless devices that have power restrictions and must reduce transmissions, or files are modified and require refreshing at a high rate. The strength of the client-based architecture is that it allows clients to protect the receipt of their content without depending on the content provider to take any additional action. Furthermore, the natural distribution of load obviates the need for advanced hotspot detection methods. The strength of both approaches is that they can be applied within the network at the same time without conflict, further strengthening the network's resistance to hotspot occurrences.

In the following sections, we consider in more detail how to design and use these daemons and plugins so that they can detect and alleviate network hotspots. We outline approaches to those questions that we believe are most critical for successful deployment of the hotspot rescue service.

3 Hotspot Early Detection

The success of the server-based rescue service depends on designing efficient and accurate algorithms for advance hotspot detection. Their efficiency is required to avoid excessive server and network overhead and accuracy is essential for avoiding false alarms, which may cause unnecessary usage of networking resources and be a potential source of new hotspots. Early detection needs to provide enough time for replicating popular objects and, in general, starting the rescue operation. However, warnings that are too early will likely increase the number of false alarms. There is another tradeoff to consider when choosing the accuracy of the server load estimation algorithms. Algorithms that yield more accurate estimates require more pro-

cessing and utilize more of the very resource the rescue service is trying to preserve. Given the complexity of optimization constraints, a comprehensive study of the interplay between analytical and experimental techniques will be necessary to discover effective advance detection mechanisms.

From the client's standpoint, the most indicative measures of the transfer quality-of-service of an object are: 1) request blocking probability: blocking typically occurs when the queue for the incoming requests fills up. 2) transfer blocking probability: this blocking event takes place when the ongoing transfer is interrupted or its service quality drops below an acceptable level; this commonly arises in streaming media applications. 3) transfer delay: the amount of time necessary to complete a transfer.

From the server's perspective, the most natural candidates for characterizing the Web server temperature are the number of new requests that are waiting for connection and the amount of unfinished work at the server. Unfortunately, both of these quantities may be difficult to measure directly without kernel modifications. Since requiring kernel modifications would complicate deployment of the HRS, one can adopt an indirect approach that can be implemented at the user level through sampling of server logs, monitoring the number and longevity of the processes running on the server, monitoring time spent blocking on an I/O device, and remote probing between participating rescue servers. One can also leverage the measurement tools already available on each operating system and hardware platform and tailor the information to the specific needs of the HRS. These tools include hardware performance counters such as those available on Pentium processors [14], standard system calls [15], the performance monitoring objects available in Windows-based systems [16], and the proc file system available in Unix-based systems [17, 18], all of which can be readily accessed in user-level programs.

To further reduce the increase in load due to server load monitoring, one needs to adopt randomized sampling approaches that balance the processing expense against the algorithm's accuracy. The main decision parameter is selecting an appropriate sampling time scale. Randomized techniques, justified by PASTA-like (Poisson arrivals see time averages) properties of object requests by web users, can be used to increase the sampling precision. Similarly, the collection of individual delay and blocking rates from the participating clients may result in massive data sets, which will require sophisticated processing methods. In the same way, the sampling rate for probing network performance and outside server monitoring needs to be minimized to avoid a potential increase in congestion induced by monitoring efforts.

In order to achieve a high degree of success in hotspot advance prediction it is essential to understand the characteristics of the server traffic load. The existing measurements indicate that this traffic exhibits a high degree of statistical variability on multiple time scales, which is often referred to as self-similarity [19]. The primary descriptors of a Web load process are the frequency of access and the distribution of document sizes. Recent studies of Web document sizes demonstrate that their empirical distributions can be accurately matched to heavy-tailed/subexponential distributions [20, 21]. Similar observations have been made regarding variable bit rate video traffic [22]. Informally, subexponential models have the ability to capture sudden large variations in the access frequency and document sizes, which is intuitively the scenario that causes hotspots.

A statistical reference model of the server traffic load can be based on heavy-tailed infinite-server type processes, which have been demonstrated to have the flexibility to capture the high variability and dependency of measured data [23, 24, 25, 26]. The parameters of the model will be derived from available statistical data.

We postulate that the most likely server overload (hotspot) takes place either when it serves a typical number of connections downloading very large files or a very large number of connections requesting relatively small documents. The former type of hotspot can be predicted using the heavy-tailed nature of the Web document distributions. These distributions have increasing residual duration times (decreasing failure rates). Therefore, from identifying persistent (large) file transfers one can conclude that the remaining work necessary to complete these transfers is potentially even larger than the already completed work. This may give a firm, early indication of a hotspot. The latter type of hotspot can be detected by monitoring the moving averages of the object access frequencies that are available from the server logs. These trends can be extrapolated into the future by utilizing the first few derivatives. For instance, the first derivative gives

the rate of access of objects, the second derivative gives the rate at which access rates are increasing. More refined prediction methods will require deeper understanding of the object frequency access process, e.g., potential understanding of a document popularity-spreading process could be achieved by modeling it as a branching process.

We briefly address the issue of detecting overload within the network (as opposed to on the servers). Such overload is likely to occur at access points to the backbone or at peering points between ISPs [27, 28, 29]. Lack of readily available information from proprietary measurements at routers within the network makes it necessary for the HRS to employ indirect probing methods to detect network hotspots [12, 13, 8, 9, 10, 11], as well as making use of available distributed probing tools that collect network statistics [7, 30, 4]. Furthermore, the plugins at the participating clients will be designed to send automatic warning messages to other systems participating in the rescue service whenever their delay and blocking rate exceed prescribed levels. The statistical correlation technique can be applied to these distributed statistical measurements to identify the potential increase in network and/or server temperature (load). One of the main mathematical difficulties in designing correlation inference techniques stems from the fact that there may exist multiple solutions, i.e., the partial probing data may be insufficient to exactly pinpoint the hotspots. Improving the correlation inference methodology is one of the objectives of the research.

Once a parametrized test for early hotspot detection and its placement within a network are specified, one can then build, install, and test the hotspot detection system as a distributed software platform consisting of client performance measurement plugins, HIP and DSERV, and server daemons `nstatd` and `sstatd`.

4 Hotspot Rescue Paradigms

In the previous section, we laid a groundwork in which an HRS could predict or at least detect the arrival of hotspots. In this section, we address specific approaches to alleviating hotspots via replication. We consider issues of replication in the server-based approach and in the P2P approach separately.

4.1 Replication in the server-based approach

Once a decision has been made by the system to replicate an object, the next question is where to put the replica. The information gathered by the hotspot daemon or plugin that detected the hotspot is crucial in making this determination. For hotspots that overload server resources, the daemon that assesses the overload contains information on the rate of the hotspot buildup at the server. Thus, it should be possible to forecast the load that the replica server, or set of replica servers, needs to provide in the short term to prevent overloading of their own resources. For hotspots that overload network resources, the daemons that detected the overload can identify those servers whose locations will enable delivery of content around the detected bottleneck.

In the server-based collaborative approach, each server wishing to take advantage of the hotspot service dedicates a portion of its own network resources to become a unit within the formation of a *mutual aid society for hotspots (MASH)*. However, the allocation of resources must be done in a manner that does not tax the server's resources to the point of eroding its ability to service its own content. It is likely that in a system that consists of numerous MASH units, rescue resources will always be available, so long as the resources are not abused by hosts that would replicate their content when it is unnecessary to do so (i.e., when the content is not sufficiently hot). To inhibit abuses, a credit-based policy is preferable. Here, each unit is initially assigned a small set of tokens. When a server hosts another server's content, the hosting server receives additional tokens, and the server whose content is being hosted loses an equal number of tokens (these tokens can simply be transferred from the content-owning server to the hosting server). A server can "rent" space on other servers using its available set of tokens. The tokens prevent participating servers from unfairly and unnecessarily renting space on other servers. If a server's token supply runs out and it still needs the replication services of the HRS, then it must buy the additional service, perhaps with actual currency.

While the token method limits the abuse of HRS resources by a single server, it does not ensure protection of an individual server participating in the MASH from being overutilized by the HRS. Each server must be ensured that the access to additional resources gained by participating in the MASH outweighs the loss of resources that must be supplied to the MASH. What is needed is a mechanism that can be deployed at a server that limits the availability of its resources (buffer, processing) to the HRS such that the server, as well as all other servers participating in the MASH, gain from this server's inclusion within the MASH. We say that policies that implement such a mechanism are *resource bounding*. Simple resource bounding policies are probabilistic denial or threshold-based denial of requests at a server for a replicated object.

Resource bounding policies may also be necessary to protect networking resources. Consider for instance the case when a group of n servers share a common set, L , of congested links or routers on the path to a set of clients. There would be little incentive for Servers within this group to replicate each others' content, since access to the replicated content would continue to pass through L . To avoid this problem, the HRS must identify and avoid replicating objects between servers whose paths to clients share common points of congestion. We propose to solve this problem by applying techniques similar to previous work that can identify common points of congestion [12]. A useful property of the techniques that differentiates them from other work in the area [31, 13] is that they do not require the transmission points of the flows to be located near one another. Hence, it can be used to determine whether or not two servers, located at different points in the network, transmit over a common set of congested links to reach a given client. Modified versions of these techniques can be applied within the first phase of the HRS through the use of probes between HRS-participating servers. In addition, it is desirable to apply these techniques during the second phase in which the probe traffic used to identify shared points of congestion is in-band, i.e., the probes are implicitly sent as part of the ongoing data traffic communication. If a shared point of congestion is identified during this second phase, then the replicated content can be moved to an alternative server within the HRS. For small data files, connections that were established prior to the move are allowed to terminate naturally. For large data files and streaming files, one can also consider the option of migrating the transmission to a new server [32].

4.2 Replication in the P2P approach

Users will also be able to control the extent to which their machine is used by the service. Again, we are interested in developing resource bounding techniques. To provide users with a high degree of control over their interaction with the service, the rescue service enables users to specify how much of their client machine and network resources can be consumed by the service. This can be done as a combination of hard and elastic limits. A hard limit ensures that resource consumption by the service never exceeds a fixed threshold, no matter what else is running on the machine. An elastic limit ensures that resource consumption by the service scales with the machine load and never interferes with other activity being performed on the machine. With elastic limits, if the machine is idle, the rescue service can consume more resources. If the machine is busy, the rescue service is limited to consume very little resources. To provide further user incentives to allocate more resources to the service, the resource allocation of the client can be used to prioritize how the given user's requests are serviced by the directory service in conjunction with requests from other users.

The proposed P2P hotspot rescue approach provides several novel advantages over existing approaches. First, because the extent of content replication is automatically scaled based on the popularity of the content, the P2P hotspot rescue service provides a very scalable approach for relieving hotspots. Second, the content replication is automatically done as part of the normal course of serving web requests without requiring any additional distribution mechanisms. Third, unlike services such as Akamai, webmasters and web developers do not have to modify their web pages in any way, shape, or form to use the rescue service; the rescue service automatically determines what content has become hot and reroutes requests before they even reach the server. This also enables the approach to be more scalable as the hotspotting server is not involved at all in handling additional requests when it is hotspotting up. Fourth, unlike services such as Napster in which

users must painstakingly determine where to obtain the requested content, the approach considered here automatically determines this without user intervention. This avoids the hassle that Napster users now face when they end up attempting to download from a site that is too slow or even no longer available. Fifth, the service provides a highly flexible resource consumption model that allows users to tightly control the extent to which they participate in the rescue service.

4.3 Migration

To this point, we have provided a general discussion of how the HRS will alleviate hotspots without consideration of the type of object being transferred. We now consider how migration of files belonging to the large data file and stored streaming classes can alleviate oncoming hotspots. The longevity of the transfer of these files creates an additional hurdle to alleviating hotspots. Once a connection is started, it is undesirable to abort that connection prior to its completion of data transfer. However, underlying conditions such as server load or network congestion may vary with time in a manner that is beyond the control of the HRS. For instance, a server may suddenly be required to process a foreground job with high processing overhead. Another example is congestion in the network due to hotspot cross-traffic (where servers and receivers whose communications trigger the cross-traffic are perhaps in alternative regions of the network and perhaps not even participating within the HRS). Since the set of links and routers required by the server to reach the client cannot be altered, and the location of the client cannot be altered, it becomes necessary to migrate the ongoing connection to another server to maintain high performance levels.

Previous work dealing with connection migration has addressed the challenge of seamlessly transferring an ongoing connection from one server to another without significantly deteriorating the performance of the transfer from the perspective of the client [32]. What has not been addressed is the issue of identifying an appropriate server to which the connection can be transferred. We are interested in performing two studies that relate to connection migration. The first is an experimental study that measures the potential performance gains that can be achieved via migration in the current Internet. The second is the evaluation of policies that are used to migrate a connection. These policies must prevent “panic” migrations that result in rapid changes in the source of transmission, in the worst case repeatedly cycling through the same set of sources, resulting in a moving, but not an alleviation of the hotspot. On the other hand, delaying migration deteriorates performance until the migration completes.

5 Notification and Termination Phases and Additional Challenges

Due to lack of space, we can simply summarize the issues to be dealt with regarding the notification and termination phases of the HRS, as well as alternative challenges that should be met once we have addressed the preliminary issues.

- In the P2P approach, how should resource discovery be performed so as to have the service determine client and network resources available for each client machine? How should that information be used to reroute requests efficiently? In the server-based approach, how are requests directed toward replicas in the server-based approach. Will this require modifications to the server or to the DNS or both? Can one make use of application level or network level *anycast service* [33, 34]?
- When should resources be withdrawn? Are multiplicative increase, additive decrease techniques appropriate? Would withdrawal at fixed threshold levels provide a satisfactory solution?
- How should the bootstrapping protocols that instantiate HIPs or server daemons within the HRS be designed? If the number of daemons or HIPs grows large, how does one restrict or control the entities that can communicate directly with one another? How is the summary information scaled?

- How can a hotspot rescue service protect against Denial of Service (DoS) attacks? How can DoS attacks be prevented from abusing the availability of the hotspot rescue service to drain more resources?
- How well do the server-based and P2P approaches work in conjunction with one another?
- What can be done for non-stored media, i.e., dynamically generated web pages? live videos? on-line games?
- How can one update the distributed directory efficiently while maintaining consistency among the various copies?
- How does the performance of the HRS models compare against one another? How do they compare with existing content-delivery architectures?
- How can one determine which directory server in the distributed directory system to route client requests to?

6 Concluding Remarks and Broader Impact

Extremely high variability of network traffic loads inevitably leads to temporary, but severe degradations of service quality in specific localized network areas. In order to improve the service in these hotspots, we recommend a novel collaborative technology in which a self-regulating, scalable, and minimally disruptive distributed rescue system identifies and rebalances excess loads by replicating hotspot content in under-utilized network regions. The research and development plan described here embraces a broad spectrum of research disciplines, from applied mathematics to software engineering.

The design problem naturally splits into monitoring technology and distributed load balancing. The improved algorithms and new software for addressing these problems can be expected to have an impact on other areas of computing research where these questions frequently arise. Specifically, the hotspot rescue mechanisms are closely related to the operation of server farms. Better understanding of distributed network monitoring may have potential use in designing other services for improved network performance.

Working prototypes incorporating research findings can be expected to provide a service that can benefit society as a whole, but in particular, the non-profit organizations, such as schools and academic institutions that otherwise would not have been able to afford commercial content distribution companies.

References

- [1] B. Huberman and R. Lukose. Social Dilemmas and Internet Congestion. *Science*, July 1997.
- [2] I. Peterson. Circumventing Traffic Jams on the Internet. *Science News*, March 1996.
- [3] Internet Bottlenecks: The Case for Edge Delivery Services, 2000. Akamai Whitepaper.
- [4] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, and Y. Jin. An Architecture for a Global Internet Host Distance Estimation Service. In *Proceedings of IEEE INFOCOM'99*, New York, NY, March 1999.
- [5] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. On the Placement of Internet Instrumentation. In *Proceedings of IEEE INFOCOM'00*, Tel-Aviv, Israel, March 2000.
- [6] S. Jamin, C. Jin, A. Kurc, D. Raz, and Y. Shavitt. Constrained Mirror Placement on the Internet. In *Proceedings of IEEE INFOCOM'01*, Anchorage, Alaska, April 2001.
- [7] S. Seshan, M. Stemm, and R. Katz. SPAND: Shared Passive Network Performance Discovery. In *Proceedings of the USITS'97*, Monterey, CA, December 1997.
- [8] R. Carter and M. Crovella. Measuring Bottleneck Link Speed in Packet-Switched Networks. Technical report, Boston University, BU-CS-96-006, February 1996.
- [9] V. Jacobson. pathchar - a tool to infer characteristics of internet paths, 1997. Presented at the Mathematical Sciences Research Institute. Available from <http://ftp.ee.lbl.gov/pathchar>.
- [10] K. Lai and M. Baker. Measuring bandwidth. In *Proceedings of IEEE INFOCOM'99*, New York, March 1999.
- [11] K. Lai and M. Baker. Measuring link bandwidths using a deterministic model of packet delay. In *Proceedings of ACM SIGCOMM'00*, Stockholm, Sweden, September 2000.
- [12] D. Rubenstein, J. Kurose, and D. Towsley. Detecting Shared Congestion of Flows Via End-to-end Measurement. In *Proceedings of ACM SIGMETRICS'00*, Santa Clara, CA, May 2000.
- [13] K. Harfoush, A. Bestavros, and J. Byers. Robust Identification of Shared Losses Using End-to-end Unicast Probes. In *Proceedings of ICNP'00*, Osaka, Japan, November 2000.
- [14] Intel architecture software developer's manual. Vol 3, System Programming Guide, Intel. <http://developer.intel.com/design/pentiumii/manuals/>.
- [15] M. J. Bach. *The Design of the UNIX Operating System*. Prentice Hall Inc., Englewood Cliffs, NJ, 1986.
- [16] M. Nielsen. *Windows 2000 Professional Advanced Configuration and Implementation*. The Coriolis Group, Scottsdale, AZ, 1999.
- [17] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom. Plan 9 from bell labs. *Computing Systems*, 8(3), 1995.
- [18] Daniel P. Bovet and Marco Cesati. *Understanding the Linux Kernel*. O'Reilly & Associates, Sebastopol, CA, 2000.
- [19] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2:1–15, 1994.

- [20] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, 1997.
- [21] P. Jelenković and P. Momčilović. Capacity regions for network multiplexers with heavy-tailed fluid on-off sources. In *INFOCOM 2001*, Anchorage, Alaska, April 2001. to appear.
- [22] P. R. Jelenković, A. A. Lazar, and N. Semret. The effect of multiple time scales and subexponentiality of MPEG video streams on queueing behavior. *IEEE Journal on Selected Areas in Communications*, 15(6):1052–1071, August 1997.
- [23] M. Parulekar and A. M. Makowski. Tail probabilities for M/G/∞ input processes (I): preliminary asymptotics. *Queueing Systems*, 27:271–296, 1997.
- [24] Z. Liu, P. Nain, D. Towsley, and Z.-L. Zhang. Asymptotic behavior of a multiplexer fed by a long-range dependent process. *Journal of Applied Probability*, 36(1):105–118, March 1999.
- [25] P. R. Jelenković and A. A. Lazar. Asymptotic results for multiplexing subexponential on-off processes. *Advances in Applied Probability*, 31(2), June 1999.
- [26] M. Krunz and A. M. Makowski. A source model for VBR video traffic based on M/G/∞ input processes. In *Proceedings of INFOCOM'98*, San Francisco, California, April 1998.
- [27] L. Gao. On Inferring Autonomous System Relationships in the Internet. In *Proceedings of IEEE GLOBECOM*, San Francisco, CA, November 2000.
- [28] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet Routing Convergence. In *Proceedings of ACM SIGCOMM'00*, Stockholm, Sweden, September 2000.
- [29] C. Labovitz, R. Wattenhofer, S. Venkatachary, and Abha Ahuja. The Impact of Internet Policy and Topology on Delayed Internet Routing Convergence. In *Proceedings of IEEE INFOCOM'01*, Anchorage, Alaska, April 2001.
- [30] Mark W. Garrett. Felix project. Telcordia Technologies, <http://govt.argreenhouse.com/felix>.
- [31] R. Caceres, N. Duffield, J. Horowitz, and D. Towsley. Multicast-Based Inference of Network-Internal Characteristics: Accuracy of Packet Loss Estimation. *Transactions on Information Theory*, November 1999.
- [32] A. Snoeren, D. Andersen, and H. Balakrishnan. Fine-Grained Failover Using Connection Migration. In *Proceedings of the 3rd Usenix USITS'01*, San Francisco, CA, March 2001.
- [33] D. Katabi and J. Wroclawski. A framework for scalable global ip-anycast (gia). In *Proceedings of ACM SIGCOMM'00*, Stockholm, Sweden, September 2000.
- [34] E. Zegura, M. Ammar, Z. Fei, and S. Bhattacharjee. Application-layer anycasting: A server selection architecture and use in a replicated web service. *IEEE/ACM Transactions on Networking*, 2000.