**MIT 18.996: Topic in TCS: Internet Research Problems** **Spring 2002**

# Lecture 1 — February 6, 2002

*Lecturer: Tom Leighton* *Scribes: Omar Aftab, Eamon Walsh*

## 1.1 Introduction

This class will discuss several research problems that are related to the Internet. Each lecture will discuss

- How a particular Internet component or technology works today
- Issues and problems with the current technology
- Potential new lines of research
- Formulation of concrete problems and solutions

We start by discussing how the Web works today, what the issues are with its current architecture, and how Akamai's services are changing it. We then discuss technological challenges, and conclude with future lines of research in this area.

Most lectures will include an example of a theoretical problem that came out of research at Akamai. Today's example, which we will see later, concerns cost optimization: the problem of assigning each user to an optimal server, whilst minimizing total bandwidth costs.

## 1.2 How the Web Works Today

The Web, when viewed from the outside, simply connects content providers to end users. The great thing about the Web is that a worldwide audience can be connected to content without licensing fees of any sort. Anyone can put up a web-site, and hundreds of millions of people can see it. This is unprecedented in human history.

While the Internet provides unprecedented connectivity, it also suffers from instability and unpredictable performance. And as the number of sites and users increase, these problems will grow dramatically.

### 1.2.1 Web Architecture: A Network of Networks

The Internet is a network of networks: today, it consists of over 9000 individual networks, that communicate using the IP protocol. These networks include large, Tier I networks, such as UUnet and PSINet, as well as numerous small ISPs. In order for the Internet to act as a truly global network connecting everyone, these individual networks must be connected together using links called 'peering' points.

A peering point is essentially a link between two routers located on different networks. For data to pass from one network to another, it must traverse through this link. In fact, there are many thousands of such peering points on the Web and in order for data to reach the end user, it must pass through many individual networks and peering points.

Two different kinds of protocols help direct traffic on the Web today. Interior Gateway Protocols, such as RIP, route data *within* an individual network. Of more concern to us is the Exterior Gateway Protocol, BGP, which is used to route data *between* networks. While interior gateway protocols use a wide variety of sophisticated methods to determine the optimal routing path – including topology, bandwidth, and congestion – BGP does not. Instead, BGP determines routes by simply minimizing the number of individual networks the data must traverse. This approach, while scalable, does not handle congestion well.

The current architecture of the Internet greatly enhances the Web's scalability and has allowed it to expand rapidly. However, inherent within this architecture are four distinct bottlenecks, that can severely degrade performance as the Web continues to grow. These bottlenecks are represented in the figure below (Figure 1.1), and we shall discuss each in turn.

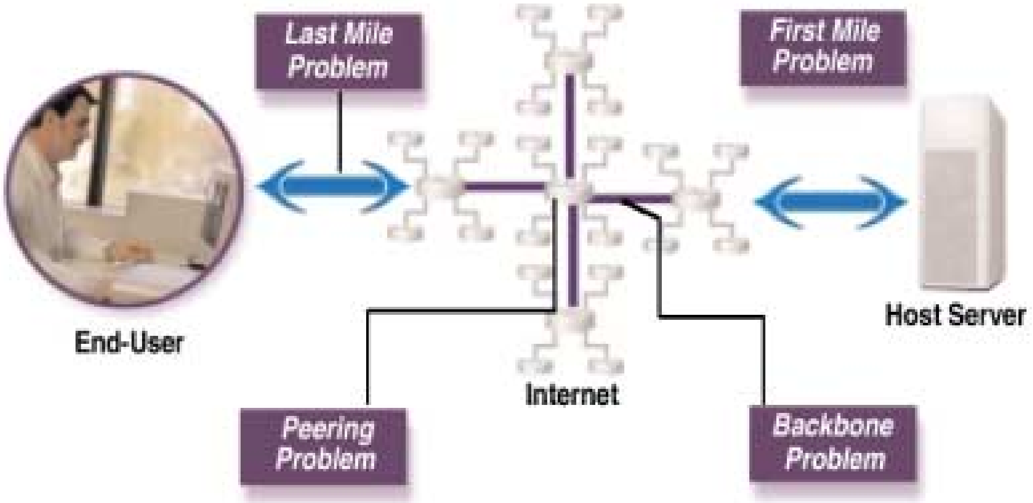


**Figure 1.1.** Problems with Internet architecture today

### The First Mile Bottleneck

The first bottleneck is a direct consequence of the fact that with 400 million users, centralization simply doesn't work. Content connected to the web at one point can quickly become swamped when demanded by a large worldwide audience. Traditionally, each content provider sets up a central web site at a single location, which provides data to the entire world. This means that *First Mile* connectivity – the bandwidth capacity of the central site – will limit performance.

In order to accommodate an increasing number of users, the content provider must continually purchase more and more bandwidth from its ISP, but more importantly, the ISP must also continually expand its *own* network capacity – and so must its peers! Thus, the centralized solution is clearly not scalable.

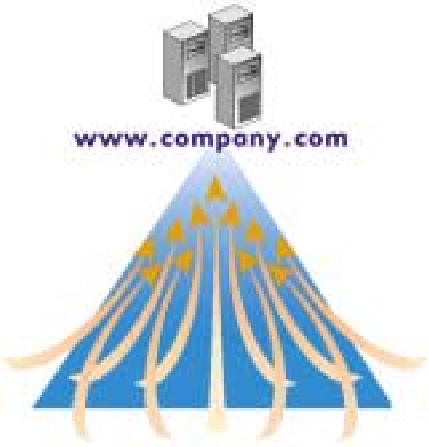


**Figure 1.2.** The first mile bottleneck

An example of this problem is the Victoria's Secret Online Fashion Show, which was heavily advertised at the 1999 Super Bowl. When more than a million subscribers attempted to access the live webcast simultaneously, the central servers could not cope, swamping the site and leaving nearly all viewers unable to view the webcast. Other examples include the non-Akamai sites serving the Star Wars trailers, and the NASA site delivering content on the Mars Rover. In each case, the demand for content was more than the first mile capacity of the Web site, swamping performance.

### The Peering Problem

The Internet today is a network of networks, with peering points between them. (Figure 1.2).

To traverse the Internet, traffic must pass through many peering points between these individual networks. Even if the bandwidth of the two networks is internally large, the bottleneck is often the access point between the two.

There is a common misconception that "big players" control large chunks of Internet traffic. If one network were to account for most of Internet traffic, most destinations could be reached within that network and no peering points would need to be traversed. However this is not the case. In reality, no single network controls more than 6% of Internet traffic [1]. Since traffic is so evenly spread out over thousands of networks, most of it must travel

[1] By traffic, we are referring to the number of bits transferred rather than the number of users. A large number of subscribers may only produce a small amount of traffic if their connections are slow, as is the case with AOL. The broadband service, Excite@Home, for example, produces about the same Internet traffic as AOL, even though it has far fewer subscribers

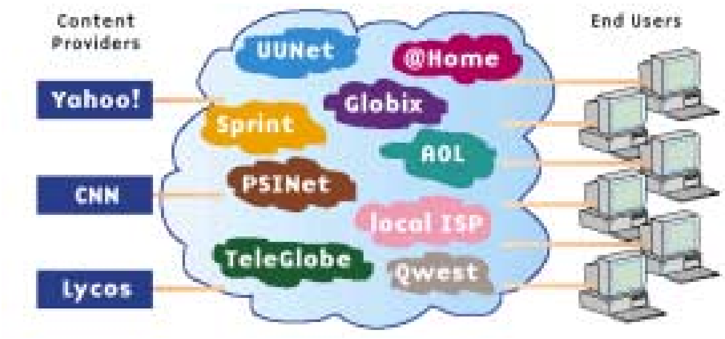


**Figure 1.3.** Many networks within the Internet

through many different networks, and consequently, numerous peering points – which are a source of congestion.

There are three main issues with inter-network peering relationships:

**Economics**

Many ISP's deliberately restrict their peering points to keep out large amounts of foreign traffic. Arguments over payment frequently occur; big ISP's traditionally charge smaller ISP's for using their backbones, but large "Tier 1" companies have traditionally not charged one another. Since the smaller networks must pay the larger ones for using their backbone, they tend to purchase only just enough capacity to handle current traffic. This means that the peering links operate at near-full capacity, resulting in packet loss and delays. Larger networks, driver by similar economic reasons, tend not to want to link with other large networks, since they aren't paid for it.

The forces of economics occasionally cause the system to break down. Cable & Wireless once denied service to several networks, demanding payment. Among them was PSINet, another Tier 1 backbone, which refused to pay. This action fragmented the Internet for three days; PSINet subscribers could not reach content hosted on C&W and vice versa. Service was only restored when the C&W hosting division was swamped with complaints from customers whose sites were inaccessible to PSINet customers.

**Routing Protocols**

As we discussed, Exterior Gateway protocols such as BGP use traditional shortest-path algorithms to determine optimal routes, ignoring congestion – even though information such as queue sizes is available to routers. As a result, when peering links get congested BGP continues to send them traffic, exacerbating the problem

**Human Error**

Routing protocols also receive instructions from human operators, which can lead to accidental loss of routes or introduction of incorrect routes. For example, systems operators are responsible for setting the number of hops to other networks. Games are often played where hops are falsely advertised in order to divert traffic onto competing networks and the like. This can lead to disaster, as in the case where a Level 3 Engineer accidentally advertised routes with a cost of $-\infty$ to all points. A tremendous amount of traffic was thus routed through Level 3, creating immense congestion. As mentioned, BGP ignores congestion in determining optimal routes, and the problem soon affected three fourths of all Internet traffic. The outage lasted for 45 minutes.

Peering, thus, is an inherent problem with the architecture of the Web. While it has been argued that peering problems should subside as telecom companies consolidate and networks merge together, there are no signs of this happening. The number of distinct networks is actually on the increase.

#### Internet Backbone

Another bottleneck is the capacity of the large, long distance networks that comprise the Internet backbone. In the traditional centralized model of web page serving, almost all requests end up traversing a backbone network from the end-user to the central server. This means that core networks must handle *all* the Internet traffic on the Web today. Currently, this is only an issue for foreign countries and global access. As Internet traffic grows, however, it may become a more pressing issue.

#### The Last Mile

The last mile bottleneck is the one that most Internet users are familiar with: the limited speed of the 56K modem link to their ISP. A common misconception, however is that the introduction of broadband DSL or cable will solve all Internet performance issues. This is not the case. Upstream issues will still cause performance problems: the connection is only as good as its weakest link. In fact, the limited speed of modems is actually saving the Internet from completely breaking down: if all users were able to demand content at broadband rates they would create so much congestion on the other three kinds of bottlenecks that Internet traffic would come to a standstill. Indeed, the Internet today is being run at close to capacity: the other issues must be dealt with before subjecting the architecture to millions of additional broadband users.

### 1.2.2 Bottleneck Implications

The current architecture of the Internet requires all traffic to pass through multiple networks to central content providers. In doing so it must encounter all four types of bottlenecks before reaching its destination. This has the following serious implications:

**Slow Downloads**
Content must traverse multiple backbones, oft-congested peering points, and long distances to reach the end user. Performance, thus is often slow.

**Unreliable Performance**
Content may often be blocked as a result of congestion or by peering problems. A site which works perfectly one minute may be inaccessible the next.

**Lack of Scalability**

Since each user must retrieve data directly from a central server, there are serious issues of scalability. Usage is limited by the bandwidth present at the central site.

**Inferior Streaming Quality**

Multimedia streaming, which is a major goal of content providers, is infeasible under the central server model. Packet loss, congestion, and narrow pipes serve to degrade stream quality, making hi-resolution video-on-demand a nearly impossible task.

**Broadband No Silver Bullet**

As we discussed, broadband is not a solution to the bottlenecks mentioned – in fact it will only serve to exacerbate first mile congestion, peering issues and backbone congestion as more and more people attempt larger downloads.

## 1.3 Akamai's Solution: Delivering On The Edge

Akamai's solution is to avoid the internet bottlenecks by providing content directly from *their* servers to the end user over the last mile. This alternative to centralized distribution is called *edge delivery*. In this model, the content of each web-site is available from servers located at the edge of the Internet: ideally, a user will be able to find all requested content on an Akamai server located on their home network. With nearly 15,000 servers distributed in over 60 countries throughout the world, the chances are good that any Internet user has an Akamai server close by.

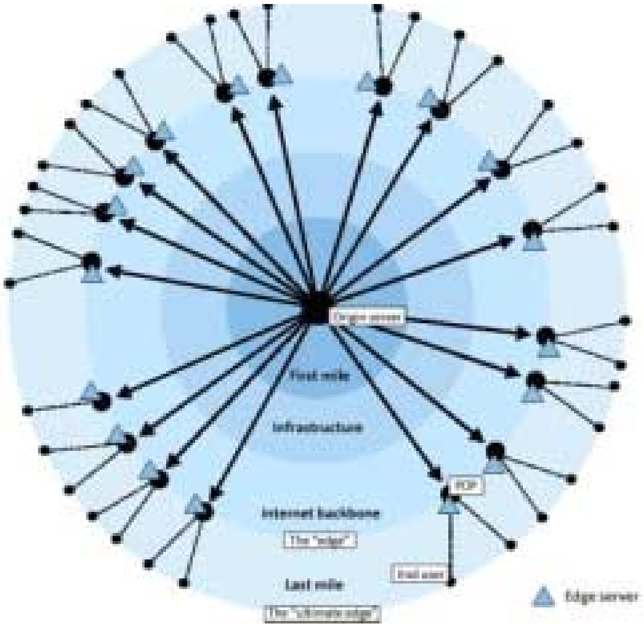


**Figure 1.4.** Edge delivery

### 1.3.1 Advantages

Akamai's model has a number of distinct advantages over the status quo. Akamai's solution avoids the first mile bottleneck by eliminating the central point from which all data was accessed. Content is served from locations close to the end user rather than from a geographically distant and oft congested central server, as shown in Figure 1.4. Since each site's content is now available at multiple servers, its capacity is no longer limited by the speed of single network link. The system is also more reliable, as there is no single point of failure. Should an Akamai server fail, its users are automatically directed to another.

Edge delivery also addresses the peering problem. Data no longer needs to pass through multiple networks and congested peering points. It can be retrieved directly from the home network, resulting in faster downloads. Of course, this means that each, or almost each, network should have its local Akamai server. Additionally, since content is delivered from the network edge, the demand for backbone capacity is also greatly reduced.

Edge delivery, of course, does not directly address the last mile problem. But by delivering content closer to end users and addressing the three upstream issues, it enables the successful introduction of broadband.

The performance improvement afforded by Akamai can be seen in Figure 1.5. The graph shows average global access times for a web-site with and without Akamai. First note that access times are significantly higher during business hours on weekdays, due to load from high-bandwidth corporate LAN's. It is clear that hosting a site with Akamai has two distinct advantages. First, the average overall access time is reduced: in some cases, up to 800%. Second, web-site performance is much more consistent throughout the week.

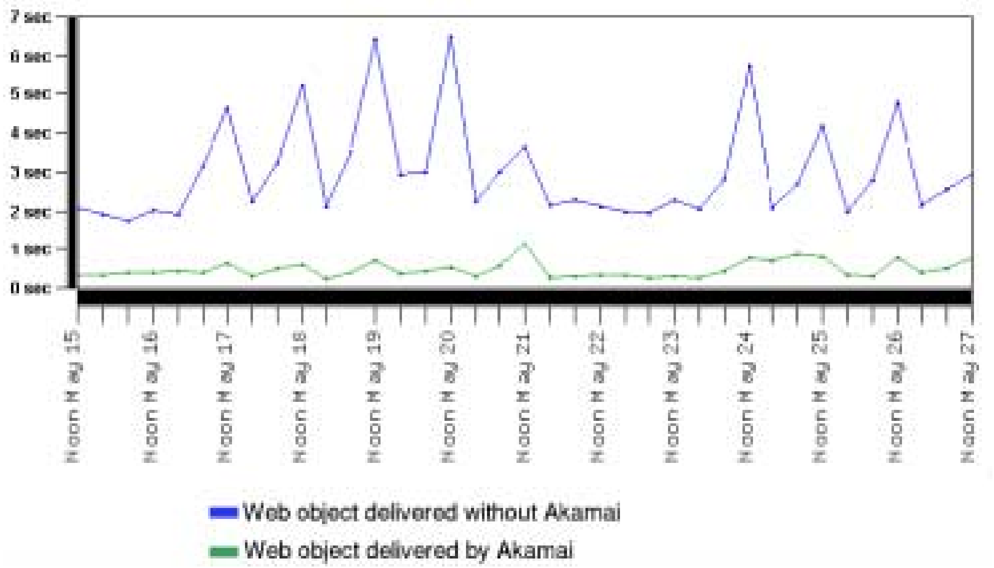


**Figure 1.5.** Web Site Performance: Typical Improvements with Akamai

### 1.3.2 Akamai's Service Offerings

- FreeFlow was Akamai's original Content Delivery Service for static web objects, such as banners and images. Like EdgeSuite, Akamai's current flagship product, FreeFlow serves content to the end user from Akamai's servers distributed at the network edge. It works by modifying the site's original HTML code to redirect the end user to an appropriate Akamai server, using sophisticated network algorithms that take into account congestion and network topology. The Akamai server then provides most of the data required to download the page. This drastically reduces communication between the end user and the content-provider's central server, thus ensuring fast downloads.

- FreeFlow Streaming uses the advantages of edge delivery to provide streaming content to viewers worldwide, with dramatic improvements in quality and reliability. For on-demand streaming, Akamai stores copies of media broadcasts on its edge servers, improving performance by ensuring that the streams are delivered to end-users from servers close to them. To support live streaming, Akamai transmits multiple streams to its network of media servers located on the edge of Web. The event is then reassembled at the edge server, and rebroadcast directly to the end user – avoiding issues of transmission quality and congestion. Akamai also uses this technology to offer streaming applications, such as Akamai Conference and Forum.

- Akamai Conference uses streaming media to extend the reach and functionality of the standard conference call. It provides live audio and video streaming, and other interactive features.

- Akamai Forum is a solution that handles the entire process of producing and broadcasting streaming media events. It enables businesses to produce live, interactive webcasts. The forum requires no special client-side software, allows live audience feedback and participation, and provides synchronized presentation slides.

- FirstPoint is a global traffic management service for content providers with multiple mirror sites. FirstPoint monitors network traffic and congestion, and connects end users to the best mirror site using DNS. FirstPoint is designed to interoperate with other Akamai services.

- EdgeScape allows content providers to customize their site content based on connection bandwidth, the user's IP address, and his geographic location. For example, this allows sites to automatically adjust their content to the bandwidth available to the user, advertise local products and services, and eliminates the need for the user to specify his location. The Edgescape system consists of local databases on servers throughout the Akamai network. These servers customize the web-pages based on the data collected about the user: again content delivery is on the edge – the servers handle user requests directly, and only contact the content-provider's central server to update their databases.

- Digital Parcel Service: Many providers today are concerned about putting content online for fear of its being utilized without due payment. DPS allows only authorized and paid users to access online material. It is a comprehensive digital distribution, reporting, and rights management solution.

- Reporter & Traffic Analyzer provide historical and real-time website usage data. These powerful tools allow customized data mining and real-time viewing of customer traffic. This allows the customer to gauge the efficacy of their advertising and marketing promotions, the popularity of their streaming events etc.

- Akamai Content Storage: is a service which allows customers to store files on strategically located Akamai servers. These files are then delivered to the end user using the Digital Parcel Service, FreeFlow, FreeFlow Streaming etc.

- EdgeSuite is the next-generation of edge delivery service; most of Akamai's services are packaged and incorporated into EdgeSuite. Launched in 2001, EdgeSuite provides content assembly, delivery, and management services from the edge of the network. It allows an edge server to assemble and serve pages from a set of variable elements, uniquely targeting each page to the individual requesting the data. As sites become increasingly customized and interactive, the ability to assemble webpages on the edge of the network without having to continually refer to the central server becomes increasingly important: it relieves load on the central content provider, and improves performance for the end-user.

  With earlier content delivery solutions, such as FreeFlow, all application processes took place at the central server: the servers at the core executed applications and assembled HTML for transmission to the end user. The speedup was gained by retrieving larger, cumbersome objects, such as graphics and media from the edge servers.

  Edge assembly content delivery solutions, in contrast, allow most of an Internet application to be cached and served from the edge, allowing for dynamic customization, personalization, and even faster performance. Using edge assembly, the server caches the results of database queries, which greatly reduces load on the central servers. Consider, for example, a site serving weather information. Such information can easily be cached for up to 15 minutes without becoming obsolete. This time, after which cached data becomes invalid and must be refreshed, is referred to as the "Time To Live". When a user requests the weather in Boston, the content will be retrieved from the central server and cached at the edge server that responded to his request. When another user requests the weather in New York, that content will also be cached. Should a third user now request the weather in either New York or Boston, the edge server will not have to contact the central server, but can assemble an appropriate page and respond directly. With many users, this technique can greatly reduce the load on the central server. Only when the TTL has expired will the the edge server need to contact the central database.

  Like FreeFlow did, EdgeSuite improves web-site performance by serving content from the edge of the network. Using EdgeSuite, however, far more content is assembled and delivered from the edge, providing even better performance.

## 1.4 Technology Overview

We will provide a broad overview of how the Akamai system works. We will be considering the details of the technology, and the issues associated with it in subsequent lectures

We begin by comparing how a site is accessed with, and without the benefit of Akamai's services.

### 1.4.1 Downloading a Website the Traditional Way

When a user requests a website such as `web.mit.edu`, a series of events take place behind the browser screen. The browser must first resolve the IP address for `web.mit.edu`. This resolution is accomplished via the Domain Name Service (DNS), which functions as the "white pages" of the Internet. At a high level, DNS maps the familiar address `web.mit.edu` to the IP address `18.7.21.77`. After the DNS service returns the IP address, the browser contacts the server at that address and requests the page. The web server returns HTML, which itself includes embedded links to other objects such as images. The browser must then repeat the lookup process for each embedded object, request it, and retrieve it. In practice, browsers may open multiple simultaneous connections to retrieve entire pages – objects, such as images, are often seen appearing on the page in the order they are retrieved.

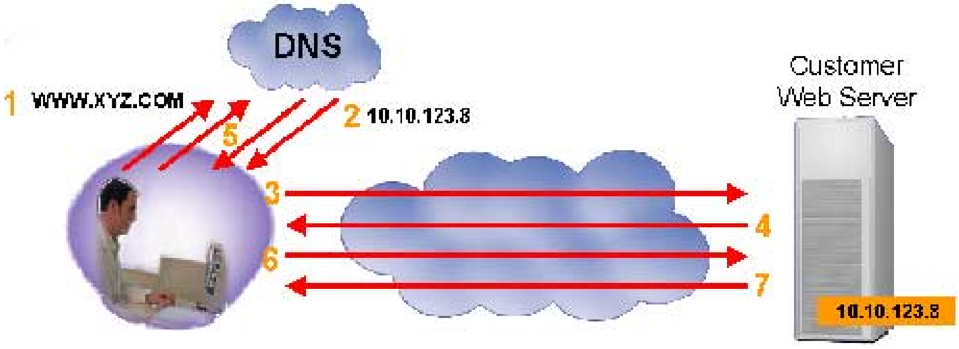


**Figure 1.6.** Retrieving a web page the old way.

1. DNS lookup for `www.xyz.com`
2. IP address for `www.xyz.com` returned
3. Browser requests HTML from server at IP address `10.10.123.8`
4. Server returns HTML, including embedded links
5. Browser performs DNS lookups for embedded objects
6. Browser requests embedded object from server
7. Server returns embedded object to browser

### 1.4.2 DNS Lookups the Traditional Way

The DNS lookup process itself consists of a number of steps. It is important to understand this process because Akamai's EdgeSuite system works by modifying it.

When confronted with an address such as `web.mit.edu`, the browser first consults its own cache to determine if it has previously resolved the name. If not, the OS, which also maintains a cache, is consulted. If this too fails, the browser must make a connection to its local name server and request an IP address for `mit.edu`.

The local nameserver maintains a record of addresses it has previously resolved. If `mit.edu` is one of them (and the entry has not expired), it simply returns the local record. Otherwise, the local name server must look to a higher authority by performing a recursive lookup. The query may eventually reach the highest authority: a root DNS server maintained by InterNIC. The InterNIC server maintains a record for every registered domain name on the web, consequently it resolves `mit.edu`. This resolution provides the IP address for the DNS server for the `mit.edu` domain.

The local DNS server now contacts the `mit.edu` DNS server to obtain the resolution for `web.mit.edu`. It receives an IP address, which it may store in its cache and return to the operating system of the original machine, which passes it, finally, back to the browser.

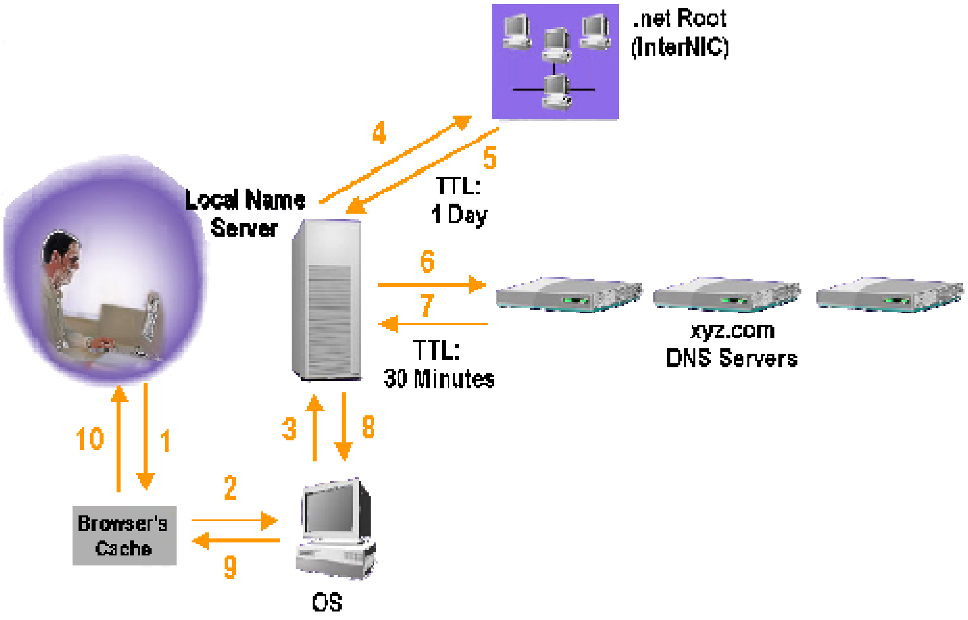


**Figure 1.7.** Doing a DNS lookup the old way.

1. Browser checks for `www.xyz.com` in its cache
2. Query goes to OS cache
3. Local name server is contacted

4. Local name server makes recursive call, eventually terminating at a root nameserver
5. Root nameserver returns IP address for `xyz.com` DNS server
6. Local name server contacts `xyz.com` DNS server
7. `xyz.com` DNS server returns IP address for `www.xyz.com`
8. IP address returned to operating system of user machine
9. IP address returned to browser, which updates its cache
10. HTML request begins

### 1.4.3 Downloading a Website the Akamai Way

When a user requests an Akamaized website such as `www.yahoo.com`, a slightly different series of events takes place. As before, the browser must first resolve the IP address using DNS. In this case, however, the address DNS returns is that of an optimal Akamai server. The browser now contacts the Akamai server to request HTML. The Akamai server assembles the webpage, making connections to the central Yahoo! server for dynamic content such as personalizations, if necessary. The HTML is then returned to the browser.

As before, this HTML may include links to embedded objects. The browser obtains the IP addresses for these embedded objects: as before, the DNS service returns the addresses for the optimal Akamai server which hosts each object. Finally, the browser retrieves the embedded objects from the optimal servers.

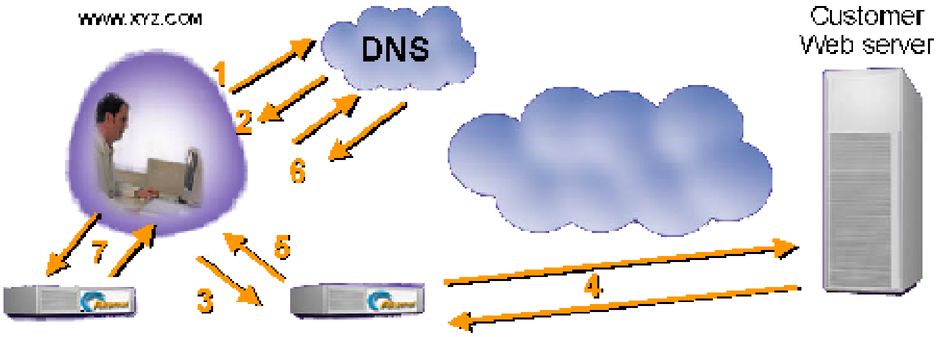


**Figure 1.8.** Retrieving a web page the Akamai way.

1. DNS lookup for `www.xyz.com`
2. IP address for optimal Akamai server returned
3. Browser requests HTML from optimal Akamai server
4. Akamai server contacts central `xyz.com` server over the Internet if necessary

5. Akamai server assembles HTML and returns to browser

6. Browser resolves embedded links, obtaining IP addresses for optimal Akamai servers hosting those objects

7. Browser retrieves objects

In the Akamai approach, there are three main issues. The DNS service must be modified to return the address of the optimal Akamai server for each particular user. Each Akamai server must be able to connect to the central server of a given website for access to database records and the like, but since this connection is made over the Internet in the traditional way, the entire gamut of issues and problems discussed previously must be dealt with. Finally, the Akamai server must assemble and deliver the page. These issues will be discussed in turn.

### 1.4.4 DNS Lookups the Akamai Way

In Akamai's approach, the DNS service must return the address of the optimal Akamai server. This requires changes to the existing DNS system. In particular, the address `www.xyz.com` must not translate directly to 18.7.21.70, but rather must be aliased to an intermediate Akamai address, for example `a212.g.akamai.net`, which will subsequently resolve to an optimal server.

Let's examine the DNS lookup process under the Akamai system. The initial stages of the lookup are exactly the same. As before, the local name server is eventually directed to the `xyz.com` DNS server. This time, however, in response to its request for `www.xyz.com`, the local nameserver receives an alias, which is known in DNS as a "CNAME." This alias is *not* an IP address, but rather is an intermediate DNS name which will resolve to the IP address of `www.xyz.com`. An example CNAME in the Akamai model would be `a212.g.akamai.net`.

The local nameserver is now confronted with the usual task of resolving a DNS address. It performs its recursive DNS query on `akamai.net`, receiving the IP address of an Akamai high-level DNS server. This server is contacted in order to resolve `g.akamai.net`. At this point, the high-level DNS server makes basic geographic calculations to determine which IP address should be resolved from `g.akamai.net`. This IP address is *not* the IP address of a web server, but rather the address of a low-level Akamai DNS server. This low-level DNS server will run a more sophisticated real-time algorithm taking into account net-congestion, local network conditions, server load etc. and determine the optimal web server for the user.

The local name server now contacts this low-level DNS server to request the resolution of `a212.g.akamai.net` and finally receives the IP address of the optimal Akamai server hosting the website content being requested.

1. Browser checks for `www.xyz.com` in its cache

2. Query goes to OS cache

3. Local name server is contacted

4. Local name server makes recursive call, eventually terminating at a root nameserver

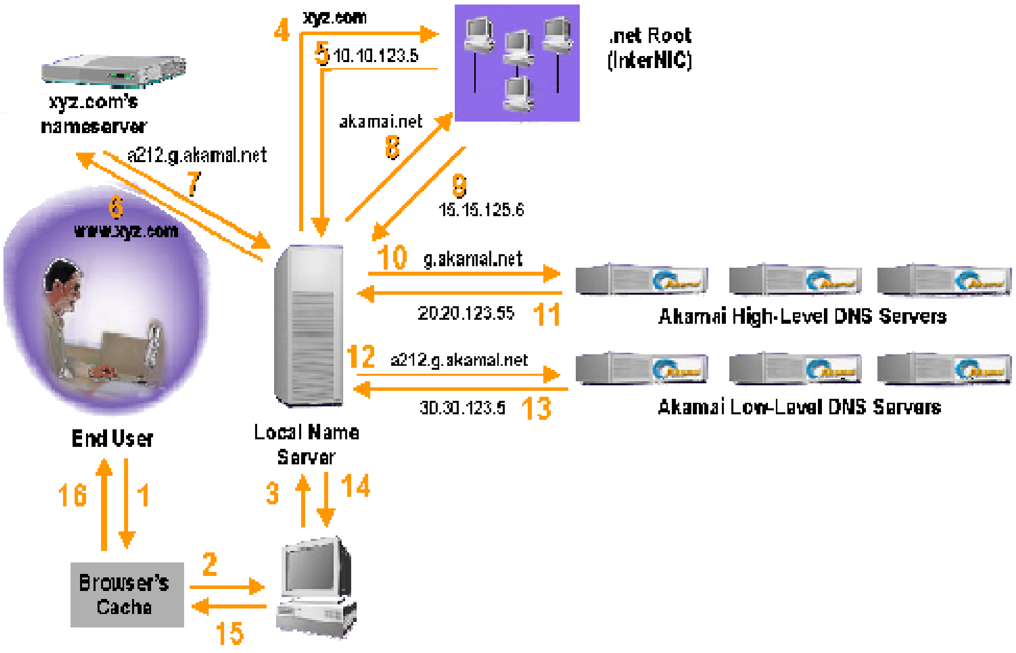


**Figure 1.9.** Doing a DNS lookup the Akamai way

5. Root nameserver returns IP address for `xyz.com` DNS server
6. Local name server contacts `xyz.com` DNS server
7. `xyz.com` DNS server returns CNAME alias `a212.g.akamai.net`
8. Local name server make recursive call to look up `akamai.net`
9. Query resolves `akamai.net` to `15.15.125.6`
10. Local name server contacts Akamai high-level DNS server at `15.15.125.6` to resolve `g.akamai.net`
11. Akamai HLDNS performs geographic calculations and refers the local name server to a geographically optimal low-level DNS server at `20.20.123.56`
12. Local name server contacts Akamai low-level DNS server to request resolution of `a212.g.akamai.net`
13. LLDNS returns IP address for the optimal hosting Akamai server
14. IP address returned to operating system of user machine
15. IP address returned to browser, which updates its cache
16. HTML request begins

### 1.4.5 Server Hierarchy & Time to Live

We mentioned earlier that cached DNS records expire after a certain “time to live” (TTL). A longer TTL means fewer recursive DNS lookups, because the cache information will be valid longer. The danger of a long TTL is that should a website’s IP address change, it may become inaccessible until the cached addresses expire.

In Akamai’s system, it is only possible to have a few high level DNS (HLDNS) servers because of InterNIC restrictions. Since all Akamai user requests must initially be handled by these servers, they are under a lot of load. If these DNS servers had to determine the optimal web-server for each user, taking into account real-time net conditions, they would become swamped. Instead, the high level DNS server makes preliminary geographic calculations and refers the user to a low level DNS (LLDNS) server, which then performs the computational intensive tasks of determining the optimal web-server. With thousands of low level DNS servers the load is thus well distributed.

A particular low level DNS server is likely to remain acceptable for a given user for at least part of a day, since geographic and high-level network parameters do not vary rapidly. Thus, when a user is directed to a particular LLDNS server, its IP address is cached for up to an hour. This further reduces the load on the HLDNS servers.

The low level DNS servers, in contrast, must determine the optimal web-server for each client, taking into account real-time conditions such as net congestion and server load *within* a geographic region. Since these are rapidly varying, once an LLDNS server directs a client to a particular web-server, the address is only cached for a few seconds. This ensures that the system responds rapidly to changing conditions; every user will be mapped to his current optimal server.

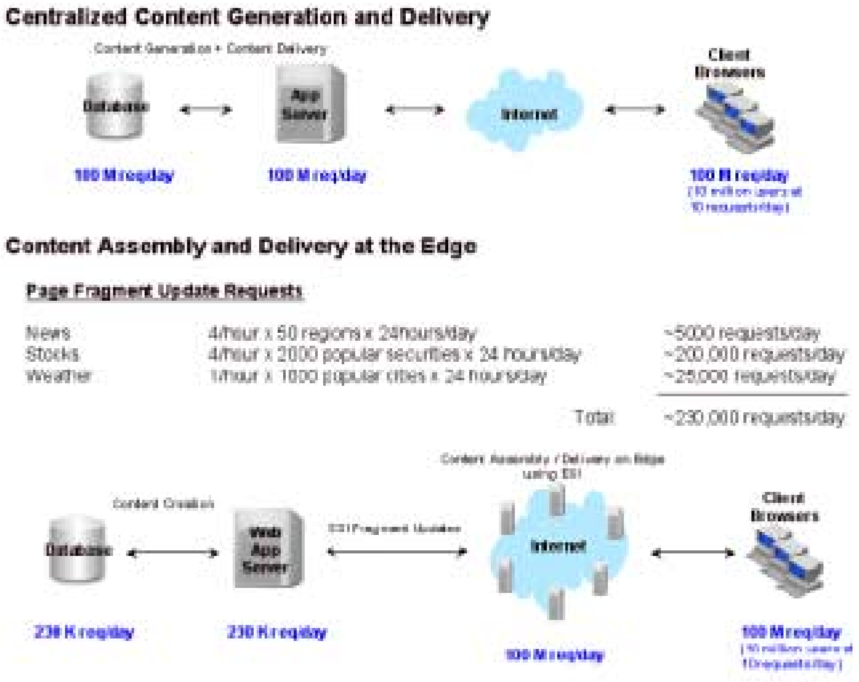
### 1.4.6 DNS Maps

The Akamai DNS servers are responsible for determining which server the user ends up receiving content from. In making these decisions, the algorithms consider geographic location, Internet congestion, system loads, server status, and user demands. The maps, created by considering all these factors, are constantly re-calculated based on the TTL’s discussed above – every hour for the HLDNS servers and every few seconds for the LLDNS servers.

### 1.4.7 Edge Assembly of Dynamic Content

As discussed earlier, Akamai’s EdgeSuite aims to generate dynamic content *on the network edge*, instead of continually referring back to the central server. Today, Web designers continue to add dynamic content – such as news, quotes, weather, customizations etc. – to their sites. In order to do so, they use Active Server Pages (ASP) or similar technologies that allow rich, personalized web site content.

However, such dynamic content poses serious problems for content delivery: the strain of delivering thousands of customized web-pages built on the fly can seriously bog down the central server – which must not only generate the content, it must also serve it. Delivering customized webpages from the network edge is also challenging, since the database remains at the central server. We have seen how Akamai’s EdgeSuite deals with this issue by caching discrete page elements called *content fragments*, and assembles them automatically based on database information to form a personalized webpage for the end user.


Centralized Content Generation and Delivery
Content Generation + Content Delivery
Database
App Server
Internet
Client Browsers
100 M req/day
100 M req/day
100 M req/day
(10 million users at 10 requests/day)
Content Assembly and Delivery at the Edge
Content Assembly / Delivery on Edge using ESI
Content Creation
Database
Web App Server
ESI Fragment Updates
Internet
Client Browsers
230 K req/day
230 K req/day
100 M req/day
100 M req/day
(10 million users at 10 requests/day)

Page Fragment Update Requests

| | | |
|---|---|---|
| News | 4/hour x 50 regions x 24hours/day | ~5000 requests/day |
| Stocks | 4/hour x 2000 popular securities x 24 hours/day | ~200,000 requests/day |
| Weather | 1/hour x 1000 popular cities x 24 hours/day | ~25,000 requests/day |
| | Total | ~230,000 requests/day |

**Figure 1.10.** Comparison of traditional server access versus edge assembly.

EdgeSuite accomplishes this by the use of "EdgeSide Includes" – an open markup language pioneered by Oracle and Akamai. ESI breaks down web pages into discrete fragments, each with its own cacheability profile. (In the context of our earlier example, the Boston weather could be one such fragment, cached with a TTL of 15 minutes.) These fragments reside on the edge servers on the Akamai networks, and are assembled into web pages at the edge servers when requested by users. The capability to assemble dynamic pages from individual fragments at the network edge means that only expired, or uncachable elements need to be fetched from the central web server. In addition, the page assembly can be conditional, based on end-user cookies or HTTP request headers. In essence, then, ESI avoids having to retrieve complete pages from the central server, greatly reducing the load it must handle.

### 1.4.8 Connections from Edge to Source

We discussed earlier that Akamai servers assemble pages for the end user. This process may require information from the central server of the site being hosted: personalized information, user preferences, etc. The Akamai server at the edge of the network thus needs to be in contact with the site's central server. In doing so, it must deal with the entire gamut of Internet connectivity issues we discussed earlier: net congestion, peering issues, etc.

How does Akamai deal with this problem? The answer lies in Routing Overlay Networks, or “Akaroutes” as they are called within Akamai. The concept is simple: use the set of nearly 15,000 geographically distributed Akamai servers to provide multiple alternate paths by connecting them together.

Rather than sending data directly from the Akamai edge server to the site’s central server over the Internet, the system considers a set of multiple paths through various Akamai servers, and choose the fastest one. For example, one way to get to site X from server A, could be to go through servers C and D, instead of connecting directly from X to A. Somewhat surprisingly, this indirect approach actually improves performance.

The reason for this is that the system maintains path performance data, and compares many possible paths to find an optimal one. It considers congestion and traffic, which the Internet as a whole does not. Even though the packets from one Akamai server to the next are sent over the Internet and are subject to the vagaries of BGP, Akamai’s complex routing algorithms ensure that the best possible hops are chosen – resulting in a fast, and reliable connection between the edge server and its sites central server.

This overlay method serves to eliminate the peering problem caused by the refusal of a given network to carry though traffic on its backbone. Since all tunneling connections are viewed as Akamai data and Akamai holds rights on all the networks it uses, those networks cannot refuse the traffic.

Finally, if a site’s central server for some reason is completely unreachable, the Akamai ACS service retrieves a default page from within the Akamai system. Thus, even if a site’s central server is down, users will still receive content at the web address.

### 1.4.9 A Note On Live Streaming

The streaming of live content such as video and audio presents its own set of challenges. As we discussed earlier, the limitations imposed by the traditional Internet are such that quality, reliable streaming is nearly impossible. Akamai’s optimized and distributed routing capabilities serve to improve conditions for live streaming by reducing the load on any given server and improving the quality of the connections delivering the stream.

Akamai employs an additional mechanism to avoid problems caused by dropped packets. Once a stream is encoded and directed into the Akamai network, it is immediately replicated and split into a series of independent substreams. These substreams are then sent over a number of different paths to localized clusters of machines which serve the stream to their local area. Only one copy of each substream is needed to reconstitute the entire stream, but since multiple copies of each substream are sent to each cluster, some of them may be lost or corrupted without preventing the local clusters from receiving the stream. This stream distribution framework, combined with the mechanisms described earlier, makes Akamai stream delivery significantly more reliable and of a higher quality.

## 1.5 Technological Challenges

The construction of the Akamai system presented several distinct technological challenges to their designers. Some of these were overcome, while others remain active areas of research.

We will now discuss some of these challenges.

### 1.5.1 Deployment & Management

For edge delivery to work, edge servers must be deployed in thousands of networks. Furthermore, these servers must be geographically distributed for optimal performance and reliability. Content must flow seamlessly from the content provider to each of these edge servers, and the service must be capable of managing content as it traverses multiple networks – this requires complex algorithms, a detailed mapping of the topology of the Web and raises complex tracking issues.

### 1.5.2 Mapping

When a user requests a webpage from the Akamai system, the system must decide which server to use. This presents a number of difficulties because of the complexity of the decision.

- One issue is simply that of scale: there are hundreds of millions of users, tens of thousands of servers, thousands of geographic locations, and thousands of host websites, thus algorithms must run in near linear time.

- Internet conditions must be constantly monitored and any changes must be *instantly* addressed to maintain optimal performance. The problem is exacerbated because Internet congestion and failures are widespread and unpredictable.

- The system must balance a load of widely varying traffic and optimize many different resources while minimizing cost.

- The system must be resilient and capable of tolerating large numbers of competent failures (as many as 1000 downed servers at once) while maintaining constant, reliable service.

- Control algorithms must be distributed throughout the network and work with imperfect information.

- DNS responses must be given within milliseconds. This is especially important because the Akamai system introduces a second level of DNS queries.

### 1.5.3 Logging, Reporting, and Billing

Another complex challenge is business-related: the logging, reporting, and billing of the more than ten billion hits per day that the Akamai system receives. The problem is especially complex because the data is distributed over 15,000 servers in 60 countries and must be retrievable in real-time for use by customers. Real-time reporting of data, real-time monitoring of performance, and real-time SQL queries must all be supported.

Akamai maintains a Network Operating Control Center (NOCC) at its headquarters to constantly monitor system-wide status. The original fault-reporting mechanisms were relatively simple, but with the increasing number of servers on the system, more and more complex reporting systems must be designed.

### 1.5.4 Operations

Yet another issue is that the huge, distributed Akamai system must always be up and running. It cannot be taken offline even for maintenance, and indeed must be capable of taking software upgrades "on the fly." Furthermore, the system must be secure against malicious attacks as well as buggy third-party software.

The difficulty of doing this is illustrated by the example of a Malaysian router which exercised an obscure bug in the Linux operating system, causing several Akamai servers to crash.

### 1.5.5 Content Freshness and Accuracy

Akamai commits to providing up-to-date content at all times. Stale content must never be served. The system must provide a way to quickly undo customer errors and refresh content. Finally, the system must provide for flexibility and relative ease of customer control over content. This is a double-edged sword, because at the same time Akamai must protect itself from customer errors, not allowing them to propagate through the system.

### 1.5.6 Management of Live Streaming and Webcasting

As webcasting and live streaming become increasingly important, the system should provide specialized options to manage them. It should be capable of utilizing and spreading information to intelligently handle packet loss. The system should optimize connection speed by constantly choosing the best path from a set of possibilities. Communication must be two-way, as interactive messaging and Q & A sessions are often desired by the customer. Data aggregation and polling are also necessary, as is the correctly synchronized delivery of audio, video, and slides together.

## 1.6 The Internet is a Triumph of Theory

We have seen an Akamai Forum presentation streamed through a Virtual Private Network (VPN) to a laptop in the classroom. Behind this technology lie a number of important algorithms. It is instructive to enumerate some of them: by doing so we realize how much impact theoretical research has had on the Web today.

1. **Network Algorithms**

   **Ethernet:** Carrier-sense multiple access

   **TCP:** Exponential backoff

   **IP:** Address hierarchy

**Minimum Spanning Tree:** Used by switches to prevent cycles in LAN's

**BGP:** Used for routing on the Internet

2. **VPN Point-to-Point Tunneling Protocol (PPTP)**

   **Encryption and Password Hashing:** Provides privacy

   **Authentication:** Confirms identity

3. **Encoding & Decoding**

   **Codec:** Encodes video and audio streams

   **Compression:** Reduces bandwidth consumption

   **Error Correcting Codes:** Improves reliability

   **Rendering:** Displays the content on screen

4. **Akamai**

   **Selection of Best Server:** Requires complex optimization

   **Billing & Reporting:** Requires real-time access to distributed data

   **Et Cetera**

## 1.7 Problem of the Day – Cost Optimization

An important research issue for Akamai in terms of profitability is cost optimization. The task is to connect each user to an appropriate server, while minimizing overall costs. Each user has a set of acceptable servers that he can be directed to, and each server has a cost associated with utilizing it.

Consider the following stylized problem: there are about a million sources of load and thousands of sinks. Each source has a set of acceptable sinks. There is also a cost per unit load associated with each sink. The problem is to sink all the loads while minimizing cost.

The simple solution is a greedy algorithm. For each source, choose the cheapest acceptable sink. This guarantees that all loads will be satisfied with minimum total cost. In our example above, the cheapest sinks for both colored sources have cost \$1. Thus the total cost is $20 \cdot \$1 + 10 \cdot \$1 = \$30$.

But now consider a more complex variant of the problem. Suppose that there are economies of scale: the cost of using a sink decreases per unit of load. Since the load here, in reality, is bandwidth, this is a more realistic assumption. The greedy algorithm we discussed before no longer works in this case, as illustrated by the following example.

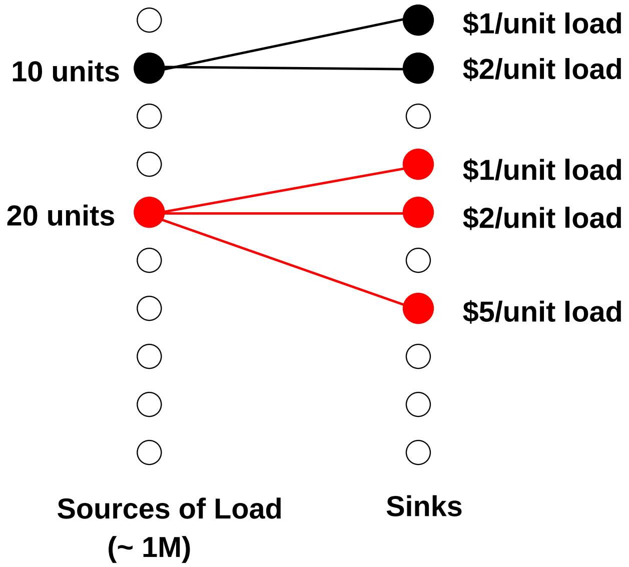


**Figure 1.11.** Simple Example

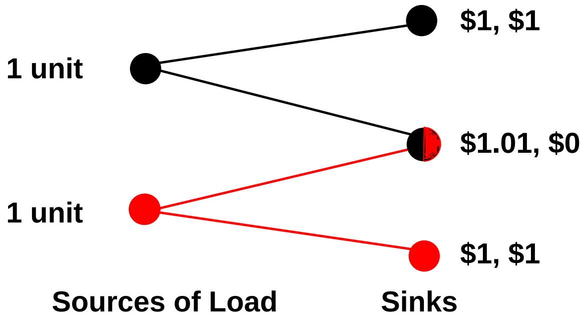


**Figure 1.12.** Greedy Algorithm Fails

The cost here is two stage. The central sink has a cost of \$1.01 for the first unit, and is *free* for every unit thereafter. The other sinks cost \$1 for *all* units. We see that the greedy algorithm selects the sinks having cost \$1 for both sources, as before. This results in a total cost of \$2. However, a cost of only \$1.01 can be achieved by assigning both sources to the center sink.

Can a feasible optimal algorithm be found to solve this cost minimization? The answer, unfortunately, is no. This problem is NP-hard, and is reducible to a k-vertex cover problem. A conceptual outline of the reduction is presented below.

### 1.7.1 The Vertex Cover Problem

Consider the graph in Figure 1.13 below. The vertices marked $B$ and $D$ in the figure have the property that they together touch all the edges in the graph. From these two vertices any other node in the graph can be reached directly. This graph is said to have a $2-$vertex covering. The $k-$vertex cover problem is that of determining whether a given graph has a set of at most $k$ vertices such that all edges in the graph touch a vertex in the set. This problem is known to be NP-complete.

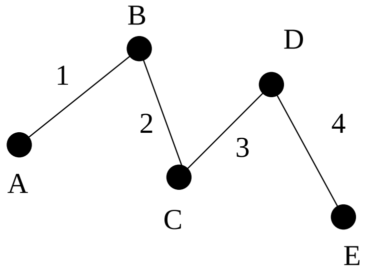


**Figure 1.13.** Example Graph

### 1.7.2 Reduction Outline

The cost minimization problem can be reduced to the $k-$vertex cover problem in the following manner. Assume we have an algorithm $A$ which, when given an instance of the cost optimization problem, returns the optimal solution in polynomial time. We will show that if this were true, then we could also solve the $k-$vertex covering problem - which is NP hard - in polynomial time.

We reduce the $k-$vertex cover problem for a graph $G$ to the cost optimization problem as follows:

1. Create a set of sources corresponding to each edge in $G$.

2. Create a set of sinks corresponding to each vertex in $G$.

3. For every sink $x$, add an edge to source $y$ if and only if the vertex corresponding to $x$ in $G$ touches the edge corresponding to $y$ in $G$.[2]

4. Set the amount of load produced by each source to 1 unit.

5. Set the cost of each sink to be \$1 for the first unit of load, and \$0 thereafter.

Call this new graph $H$. $H$ is clearly an example of the cost optimization problem as we have described it.

An example of this transformation is diagrammed in Figure 1.14 below for the example graph shown earlier.

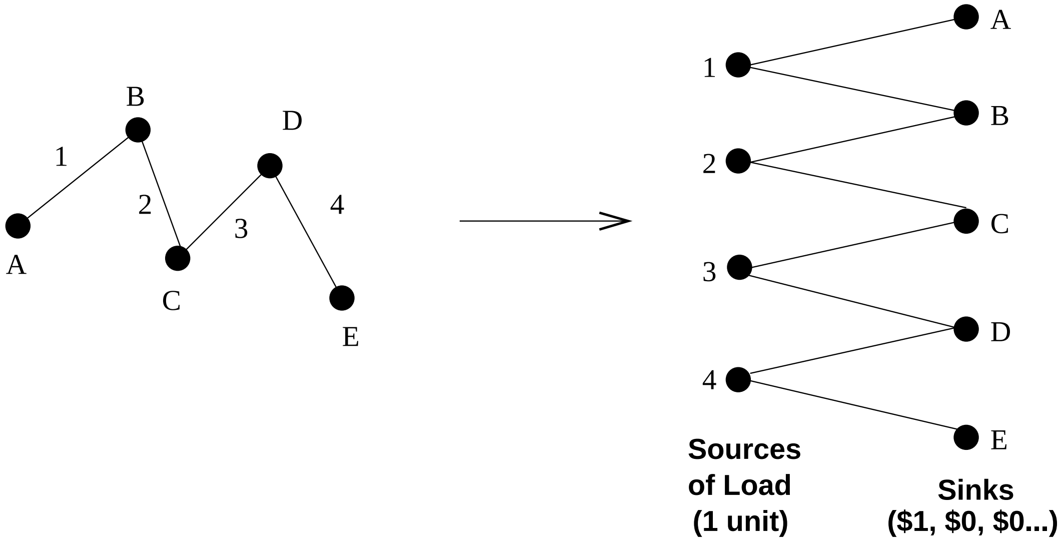


**Figure 1.14.** Example Reduction

Recall that the cost of using a sink in $H$ is \$1 for the first unit, and \$0 thereafter. Thus it is locally optimal to direct all the load from a source to a *single* sink, without loss of generality. Suppose a source has its first unit of load directed to sink A. Then all other load coming from that source can be directed to A with total additional cost 0, preserving a locally optimal solution.

Observe that if $H$ can be solved with cost $\$k$, then all the sources of $H$ must be assigned to exactly $k$ sinks. To see this, note that each sink used costs \$1 – regardless of how much load is assigned to it. Therefore, if $H$ can be solved using $\$k$, then $k$ sinks must have been used.

By definition of the cost optimization problem, all sources of load must be assigned a sink in order to form a solution. We have found that $k$ sinks are sufficient to handle all the load in $H$. Thus, these $k$ sinks must be connected to *every* source in $H$.

[2] The "acceptable set" of sinks for a source is represented here by the sinks connected to that source by edges.

Recall that each sink in $H$ corresponds to a vertex in $G$ while each source corresponds to an edge. But, if $k$ sinks are connected to every source in $H$, this means that $k$ vertices in $G$ are connected to every edge in $G$ – which means that $G$ has a $k-$vertex cover!.

Thus, we can define an algorithm $B$, which when given an instance $G$ of the $k-$vertex cover problem, constructs the corresponding cost-optimization problem using steps $1-5$ above, applies $A$ to obtain a solution, and checks to see if $k$ or less sinks are used. If so, $B$ knows that $G$ has a $k-$vertex cover. But if $A$ runs in polynomial time, then so does $B$. Since the vertex cover problem is NP-hard, no such (known) algorithm $A$ exists. This completes the reduction[3].

[3] The reverse reduction is relatively obvious. If $G$ has a k-vertex cover, this means that there is a set of $k$ vertices connected to *every* edge in $G$. Correspondingly, this means that there is a set of $k$ sinks connected to *every* source in $H$. Since each sink costs \$1 to use regardless of the load it handles, there is a solution to $H$ with cost \$$k$.

# Bibliography

[1] Akamai Whitepaper: "Internet Bottlenecks: the Case for Edge Delivery Services." 2000, Cambridge, MA. Available URL: http://www.akamai.com/en/html/services/white_paper_library.html

[2] Janiga, M; Dibner, G. & Governali. F. "Akamai & Internet Infrastructure: Content Delivery. " 2001, Goldman Sachs Equity Research.

[3] Leighton, Tom. Presentation: "The Challenges of Delivering Content on the Internet." 2002, Cambridge, MA.