

BOTTLENECK MANAGEMENT: A NEW APPROACH TO BANDWIDTH
MANAGEMENT

A thesis presented to
The Faculty of
The College of Arts and Sciences of Ohio University

In partial fulfillment
of the requirements for the degree
Master of Science

Harley A. Stenzel
August 1997

BOTTLENECK MANAGEMENT: A NEW APPROACH TO BANDWIDTH
MANAGEMENT
BY
Harley A. Stenzel

This thesis has been approved
for the Department of Math
and the College of Arts and Sciences by

Shawn D. Ostermann
Assistant Professor of Computer Science

Leslie A. Flemming
Dean, College of Arts and Sciences

ABSTRACT

Stenzel, Harley A., Ohio University, August 1997. Bottleneck Management: A New Approach to Bandwidth Management. Major Professor: Dr. Shawn Ostermann.

At most sites today, even relatively well-connected sites, a single user can overwhelm the site's network connection, and there is very little that the site can do to detect this situation, let alone correct it or prevent it from happening. This thesis describes a new approach to bandwidth management that gives the ability to detect, correct, and prevent this and other situations.

This work proposes a new paradigm for bandwidth management called integrated bottleneck management. This work also discusses the design and implementation of a prototype system for integrated bottleneck management, and makes recommendations regarding the feasibility, value, and future work on integrated bottleneck management.

ACKNOWLEDGMENTS

I'd like to thank Dr. Ostermann for his mentorship and direction and Ohio University Communications Network Services for enabling me to look carefully at this subject.

I'd also like to thank Lawrence Berkeley Laboratory and the University of California for libpcap and pcapure, as those tools were extremely useful in the implementation of the probe.

DISCARD THIS PAGE

TABLE OF CONTENTS

	Page
ABSTRACT	iii
LIST OF TABLES	viii
LIST OF FIGURES	ix
1. INTRODUCTION	1
1.1 What is Bandwidth Management?	1
1.2 Existing Models for Bandwidth Management	1
1.3 The Goals of This Work	2
1.4 Organization of Thesis	3
2. BANDWIDTH MANAGEMENT FUNDAMENTALS	4
2.1 Bandwidth Management Approaches	4
2.2 Deployed Bandwidth Management Mechanisms	6
2.2.1 RSVP	6
2.2.2 ATM	6
2.2.3 Frame Relay	7
3. A NEW APPROACH TO BANDWIDTH MANAGEMENT	9
3.1 Bottleneck Management	9
3.2 Retroactive Bottleneck Management	11
3.2.1 Congestion Avoidance Protocols	12
3.2.2 Non-Congestion Avoidance Protocols	12
3.3 An Integrated Bottleneck Management System	13
3.4 A Proposed Mechanism for Managing Bottlenecks	14

	Page
4. PROTOTYPE DESIGN	17
4.1 Design Overview	17
4.2 Design Assumptions	18
4.2.1 Router Considerations	18
4.2.2 Device Placement	18
4.3 Design Considerations and Solutions	20
4.3.1 Deployment Requirements	21
4.3.2 Scalable	22
4.3.3 Centralized and Simplified	22
4.4 Distribution of Tasks	22
4.4.1 Intelligent Agent	22
4.4.2 Probe / Observation Agent	23
4.4.3 Enforcement Agent / Router	23
5. IMPLEMENTATION	24
5.1 The Observation Agent probe	24
5.1.1 Receiving Filters	25
5.1.2 Filtering Packets	26
5.2 The Intelligent Agent intagent	29
5.2.1 The Language	29
5.2.2 Dealing with probe	31
5.2.3 Dealing with the Enforcement Agents	31
5.2.4 Notable Limitations of intagent	31
5.3 Tests and Results	31
5.3.1 Testing Probe	31
5.3.2 Testing intagent	32
6. CONCLUSIONS	34
6.1 Feasibility	34
6.2 Value	34
6.3 Future Work	35
BIBLIOGRAPHY	36

APPENDIX

A. FILTER.H	38
B. GRAMMAR	42

LIST OF TABLES

Table	Page
5.1 Probe Performance With 2000 Packets in the Circular Buffer	33

LIST OF FIGURES

Figure	Page
2.1 Connection Bandwidth Management (RSVP and ATM)	7
2.2 Segment-Orientated Bandwidth Management (Frame Relay)	8
3.1 LAN/WAN Topology with Bottlenecks Identified	10
3.2 Cooperative Bandwidth Management	11
3.3 Retroactive Bandwidth Management	12
3.4 Distributed Multi-Segment Bandwidth Management	16
4.1 Acceptable Probe Placement 1.	19
4.2 Acceptable Probe Placement 2.	19
4.3 Probe Cannot Observe Bottleneck	20
5.1 How <code>probe</code> Works	28
5.2 Example Configuration File for <code>intagent</code>	30

1. INTRODUCTION

At most sites today, even relatively well-connected sites, a single user can overwhelm the site's network connection, and there is very little that the site can do to detect this situation, let alone correct it or prevent it from happening. This thesis describes a new approach to bandwidth management that gives the ability to detect, correct, and prevent this and other situations.

1.1 What is Bandwidth Management?

Bandwidth Management is simply deciding how to use bandwidth and enforcing that decision. In circuit switched networks, such as the telephone network, bandwidth management is trivial – allocate one circuit to one connection for as long as both parties want the connection. In packet switched networks, bandwidth management is still in its infancy. There are neither widely deployed nor standardized methods for bandwidth management in packet switched networks.

1.2 Existing Models for Bandwidth Management

Currently, approaches to bandwidth management fall into two categories: those approaches which are “connection orientated” and those which are “segment based”.

“Connection orientated” are approaches that guarantee a certain amount of bandwidth to end-user applications, making virtual circuit switched networks.

The “segment based” approaches consist mainly of proprietary approaches implemented in the routers of some vendors. These are supported as a way for routers to guarantee support for different protocol families on the same wide area connection.

There is currently no approach that allows an organization to make organization wide decisions about how it will use its bandwidth.

1.3 The Goals of This Work

The goals of this work are the following.

1. To develop a framework within which an organization can make and enforce decisions about how to allocate its bandwidth. The framework must:
 - Be unilaterally deployable. An organization should be able to manage its own bandwidth without requiring any assistance from other organizations.
 - Be prudent. The framework should take into consideration the fact that it is more important to manage some segments than others, and that it is more important to manage segments where there is already a problem. This could be called “bottleneck management”.
 - Be flexible. There are many ways which an organization may want to manage its bandwidth usage. This framework should not unnecessarily limit the ways of specifying how bandwidth is to be used or actions that are to be taken.
 - Be scalable. An organization should be able to manage bandwidth on as many or as few segments as it wants, and this framework should readily scale to support a large number of managed segments.
 - Be comprehensive. The management agent must be able to look at its complete body of information about the state of the portions of the network

that it is managing, and be able to make policy enforcement decisions based on that complete body of information.

2. To test the feasibility and usefulness of such a system by building and testing a prototype.
3. To gain experience with this integrated form of bottleneck management.
4. To make recommendations about its feasibility, usefulness, and what future work, if any, should be considered.

1.4 Organization of Thesis

This thesis contains six chapters and two appendices. Chapter one is the introduction which discusses the motivation and goals for this work. Chapter two introduces the reader to bandwidth management and to the services that are already available to support this work. The proposed approach is described in chapter three. Chapter four contains the design for the prototype, the implementation of which is described in chapter five. Chapter six provides the conclusions and recommendations for further work. Finally, the first appendix contains the header source code for the prototype developed during the course of this work, and the second appendix contains the configuration file grammar.

2. BANDWIDTH MANAGEMENT FUNDAMENTALS

The currently-popular approach to bandwidth management is to build a virtual circuit switched network. This approach guarantees a certain quantifiable *quality of service (QoS)*¹ for any given connection. This approach offers some immediate advantages to the end-user of timing-critical network applications, such as guaranteeing the constant quality of audio and video over the internet. Even though this approach offers advantages to end-users, this approach does not allow a site to better manage its bandwidth bottlenecks because this approach simply gives bandwidth guarantees on a first-come first-served basis, not a high-priority low-priority basis.

2.1 Bandwidth Management Approaches

In general, bandwidth management provides a framework that allows certain amount of throughput to be reserved for a particular purpose. Within this framework, it could refer to a user's network application reserving a connection of a certain size across the internet, just as it could refer to a business deciding that traffic to its partners is more important than other traffic on a wide area network link.

Approaches to bandwidth management generally fall into two categories – those that guarantee bandwidth for a particular connection and those that guarantee bandwidth on a particular segment. The former approach is generally approached as a protocol, since a protocol is required to negotiate the terms of a specific connection.

¹Quality of Service guarantees certain parameters for a connection. These parameters can include, but are not limited to the amount of minimum bandwidth, maximum delay, maximum jitter (variation of time between packet delivery), and maximum loss rate. Parameters vary for different applications, as different applications need different service qualities: some need minimal delay and reliable response times, while others may need a good image quality.

This is the approach found in RSVP [ZDE93] and ATM [IBM94]. The latter approach is generally implemented at a router since the router is responsible for ensuring that it does not inject packets too quickly onto certain kinds of networks, such as frame-relay networks.

In order for a network application to reserve bandwidth along the entire route of a virtual circuit switched network end-to-end connection, there must be unallocated bandwidth in each segment along the route of a connection. Assuming the virtual circuit switched network grants the application's quality of service request, and the application uses the bandwidth that it was guaranteed, there must be a method of enforcing the decisions made by the virtual circuit switched network (VCSN) during the allocation, otherwise there is no guarantee that the bandwidth allocated will be available to the allocator. For this to be the case, every segment along the end-to-end path must agree to the terms of the quality of service guarantee. Furthermore, a VCSN does not even begin to address the question "How does my site want to use the connectivity that it has?" On top of that, the VCSN model breaks if even a single segment along the end-to-end path does not support the VCSN. Therefore, any virtual circuit switched network implementation requires support from every segment along every path that wants quality of service guarantees, which means that every router along the end-to-end path must support the VCSN implementation.

For a business to implement the policy "Traffic to my partners is more important than other traffic on my WAN link," there must be some agent that is able to enforce the business' policy. That agent, generally a router, can enforce policies by deciding which packets to forward and which to drop [Cis97a]. Although solutions such as this are proprietary to specific routers, they can be implemented without the support of the rest of the world. Of course, they do not guarantee to specific instances of applications that they will have the QoS that the application requests, but WAN links can be configured to allocate a certain portion of the total bandwidth for specific applications in some cases.

In both of these examples of managing bandwidth, the services that underlie each method of bandwidth management are allocation and enforcement. The allocation service deals with what is available, what is used, and who may have what. The enforcement service ensures that the way the bandwidth is used conforms to the way that it was allocated.

2.2 Deployed Bandwidth Management Mechanisms

RSVP and ATM both do connection-orientated bandwidth management. Both provide minimum quality of service guarantees to the owners of connections, while Frame Relay guarantees minimum quality of service to the owner of a site.

2.2.1 RSVP

The RSVP protocol is used by a host to request specific qualities of service from the network for particular application data streams or flows. RSVP is also used by routers to deliver quality of service requests to all nodes along the path(s) of the flows and to establish and maintain state to provide the requested service. RSVP requests will generally result in resources being reserved in each node along the data path [ZDE93]. See Figure 2.1.

2.2.2 ATM

Like RSVP, ATM is used by a host to request specific qualities of service from the network for particular application data streams or flows. ATM is also used by routers to deliver quality-of-service requests to all nodes along the path(s) of the flows and to establish and maintain state to provide the requested service. ATM requests will generally result in resources being reserved in each node along the data path [IBM94]. See Figure 2.1

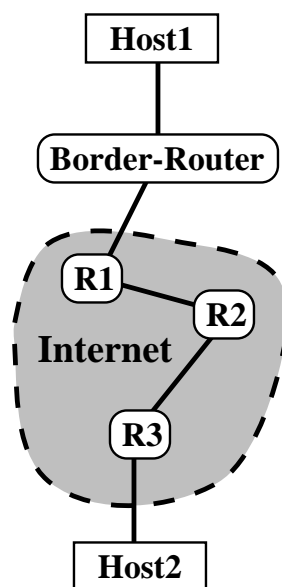


Figure 2.1 Connection-Orientated Bandwidth Management

Host 1 reserves bandwidth along a path to Host 2. Every router along the path (Border-Router, R1, R2 and R3) agrees to guarantee a certain quality of service.

2.2.3 Frame Relay

Frame Relay is a digital data circuit operating at any one of a variety of pre-subscribed speeds. An individual or an organization buys a portion of the bandwidth of a frame relay circuit from an Internet Service Provider (ISP). The router at the customer's site ensures that the customer does not use more bandwidth than their agreement allows [Lyn96]. Frame Relay does bandwidth management because without bandwidth management, each packet competes for each bit of bandwidth. Frame relay says that "everyone behind this router gets this amount of bandwidth", and therefore makes bandwidth guarantees to each site on the frame relay. See Figure 2.2.

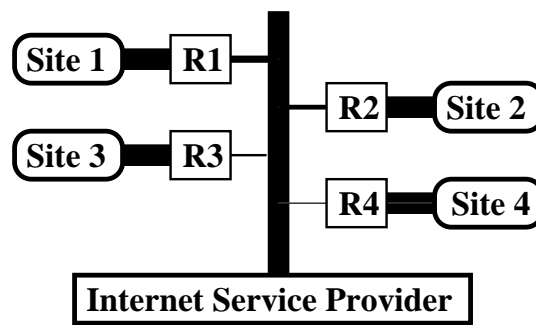


Figure 2.2 Segment-Orientated Bandwidth Management
The Internet Service Provider has agreed to provide each site at least a set amount of bandwidth. A router at each site enforces that agreement.

3. A NEW APPROACH TO BANDWIDTH MANAGEMENT

To fill a gap in an organization's ability to manage its network and its bandwidth, this chapter outlines a new approach to managing an organization's bandwidth by managing the portions of the network where there is the most contention for the bandwidth.

3.1 Bottleneck Management

An internetwork can be thought of as a combination of relatively high speed, low cost local area networks and relatively low speed, high cost wide area links. Available bandwidth on well-planned and well-maintained local area networks is usually not scarce. Additionally, internetworks are planned with the assumption that most network traffic is local. Today, with the advent and popularization of the WWW and the internet, a larger and larger portion of network traffic is non-local. Meanwhile, wide area links are still relatively expensive and relatively slow, so even more pressure is being put on every site's WAN links. That pressure on a site's WAN links is at the bottlenecks where LANs meet the WAN links. One obvious approach to managing a site's bandwidth is to manage the site's bottlenecks.

Additionally, because a relatively low bandwidth, high cost connection is used in nearly every case when a site connects to the Internet, managing bandwidth into and out of a site is simply a special case of managing bottlenecks (figure 3.1).

Ideally, to manage a bottleneck, the enforcement agent would permit or deny access into the managed segment based on the management policy. In figure 3.1, the agent at bottleneck "b" would perform this function for traffic from LAN 1 to

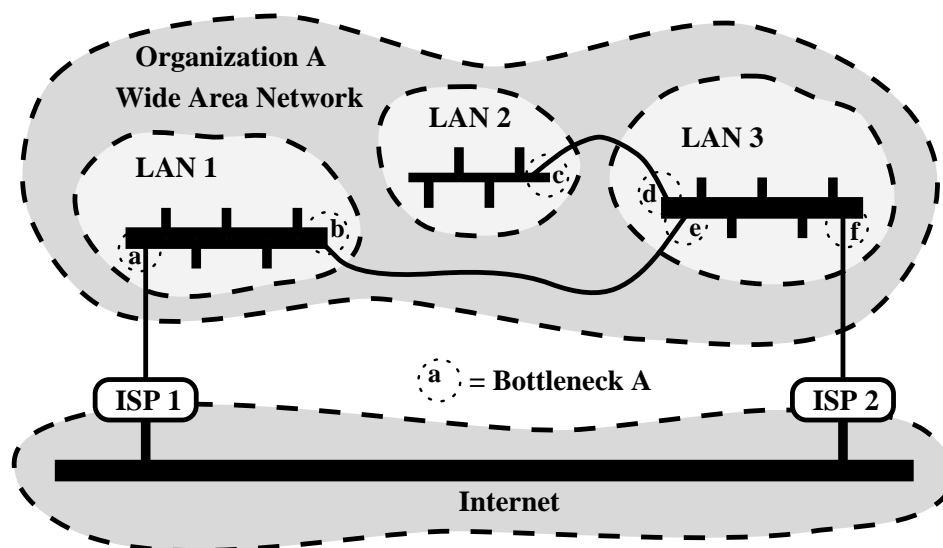


Figure 3.1 LAN/WAN Topology with Bottlenecks Identified

A multi-homed, multi-site organization identifies its bottlenecks prior to managing the bandwidth through them.

LAN 3, and the agent at bottleneck “e” would perform this function for traffic from LAN 3 to LAN 1. A similar relationship holds for traffic between LAN 2 and LAN 3. Bandwidth through the bottleneck is managed because the router on each side of the slow pipe enforces a packet forwarding policy, and through cooperation the two routers manage all of the bandwidth through the bottleneck (Figure 3.2). Furthermore, since the existing support from routers comes only in the form of limiting that which is forwarded, this approach has little overhead (Subsection 2.2.3).

At bottlenecks “a” and “f”, this scheme will not work without the cooperation of the ISPs involved. With the ISPs cooperation, the management problem is identical to the one described above. Without the ISPs cooperation, however, Organization ‘A’ has two options: either it can do nothing, or it can ‘retroactively’ manage the bottleneck.

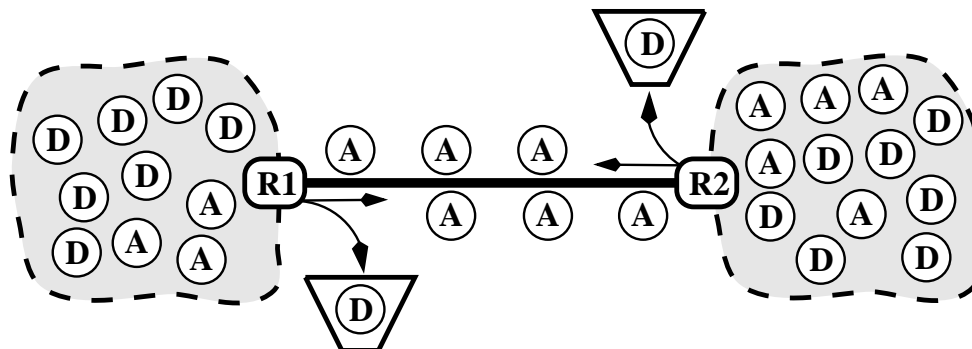


Figure 3.2 Cooperative Bandwidth Management

Two routers, acting as enforcement agents, enforce a bandwidth management policy and allow some packets to enter the WAN link while discarding others. Each router only manages outgoing packets. Packets with an “A” are allowed, while packets with a “D” are discarded and thrown into the bit bucket.

3.2 Retroactive Bottleneck Management

By the time a bandwidth enforcement agent for Organization ‘A’ has an opportunity to decide whether or not it should forward a packet through the bottleneck, the packet has already gone through the bottleneck. So, with retroactive bottleneck management, the enforcement agent decides at that point (after the packet has crossed the bottleneck) whether or not to forward the packet on to its final destination (Figure 3.3). To understand the motivation for not forwarding packets that have already crossed the bottleneck, it is necessary to look at congestion avoidance protocols and the characteristics of non congestion avoiding protocols.

Retroactive bottleneck management is more costly than other forms of bottleneck management. Because a packet is discarded after it has already consumed the limited resource, in one sense the damage has already been done. Retroactive bandwidth management is simply “complaining” about the behavior of the disallowed packets by refusing to forward them.

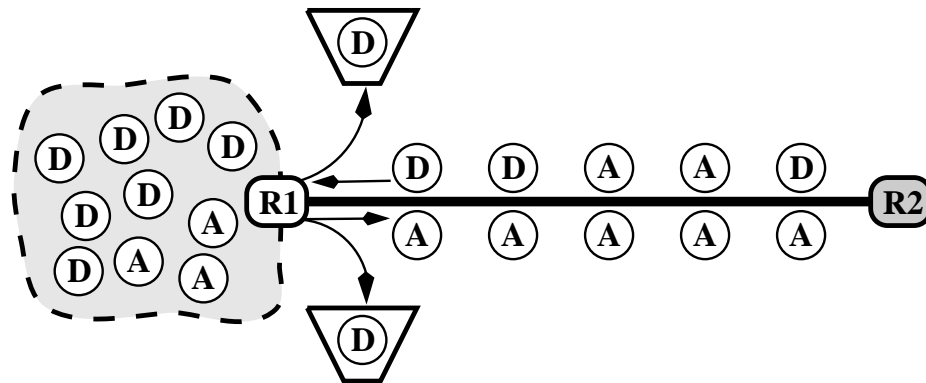


Figure 3.3 Retroactive Bandwidth Management

Router 1, acting as an enforcement agent, enforces a bandwidth management policy and allows some packets to enter the WAN link while discarding others. This router also allows some packets to leave the WAN link while discarding others, because router 2 does not help enforce the bandwidth management policy. Packets with an “A” are allowed, while packets with a “D” are discarded and thrown into the bit bucket.

3.2.1 Congestion Avoidance Protocols

In congestion avoidance protocols, causing a loss event will cause the transmitting host to slow the rate at which it introduces new packets into the network. For instance, if a packet is lost from a TCP (Transmission Control Protocol) [Soc91] [Pos81] session, when the sender detects a loss it halves its congestion window, which effectively halves the instantaneous throughput for the connection [Com88]. By not delivering a packet to a congestion avoiding protocol, the enforcement agent causes the protocol to slow the transmitter down, which is what the enforcement agent would have done directly had it been able to.

3.2.2 Non-Congestion Avoidance Protocols

In protocols that do not use congestion avoidance, retroactive bottleneck bandwidth management depends on two factors to work effectively. First, note that in

many cases, congestion avoiding protocols already have a higher priority for forwarding than on congestion avoiding protocols. This holds true in the IP family: Cisco routers forward TCP packets with a higher priority than UDP packets, and UDP packets are not forwarded in cases where the bandwidth is already consumed by TCP packets [Cis97c]. Secondly, by degrading the performance of a best-effort datagram delivery service, the user is encouraged to turn it off. Such protocols include RealAudio [Pro96] and CU-SeeMe [Sat96]. Since this service is used primarily for applications such as real time audio and video that can tolerate some (but not too much) loss, degrading the performance of this sort of traffic can directly encourage the user to “stop it”.

3.3 An Integrated Bottleneck Management System

Integrated bottleneck management is analogous to building a series of dams to protect a town from flooding. A certain amount of water can flow through a town safely, and much of the year no management is needed. But when the rains come, and the river can no longer safely carry all of the water through town, decisions about which dams should release how much water are made not by looking at a single dam, but by taking into consideration all of the dams, or the condition of the entire managed network.

By managing a site’s bandwidth, the site administrators are saying that for the wellbeing of the town it is more important that certain reservoirs be drained than others. In other words, it may be more important to one site that packets of electronic mail be forwarded than it is important that packets of web radio be forwarded.

In packet switched computer networks, the dams already exist. The dam is the capability of routers to regulate the rate at which they introduce new packets into a network segment. What is missing is the knowledge about which network segments

are bottlenecks, the ability to detect “flood stage”, or congestion, and the knowledge of what to do when “flood stage” is detected.

Integrated bottleneck management system is composed of four components.

- **Configuration**

“What does the management system have to know about the network?” and
“Where are its detection and enforcement agents?”

- **Detection**

“What conditions should be monitored and who should be told if something goes wrong?”

- **Decision**

“When conditions are detected, what actions should be taken?”

- **Enforcement**

“How will those decisions be carried out?”

3.4 A Proposed Mechanism for Managing Bottlenecks

The management system proposed here has three pieces: an intelligent agent, an observation agent (or probe), and an enforcement agent. The interaction between these agents is described in figure 3.4.

The intelligent agent is responsible for the **Description** and **Decision** dimensions for an integrated bottleneck management system described above. The management tool user is responsible for describing the network and management resources as well as the network management policy to the intelligent agent. The intelligent agent is then responsible for communicating to the probe the conditions that the probe is monitoring for, and for communicating the intelligent agent’s decisions to the enforcement agent. None of the deployed approaches to bandwidth management have a piece that is analogous to the intelligent agent.

The observation agent, or probe, is responsible for **Detection**. It watches network traffic on the bottleneck, accepts parameters from the intelligent agent, and reports back to the intelligent agent when the conditions that the intelligent agent requested the probe watch for are satisfied. There are currently widely deployed and standardized network monitoring devices that are capable of doing many, but not all, of the functions of the observation agent [Wal91] [Wal93] [Wal95].

The **Enforcement** agent is responsible for accepting management policies from the intelligent agent, and ensuring that at the specific bottleneck that each enforcement agent guards, that policy is carried out. There already exists the ability in some routers to guarantee qualities of service for various protocols, networks, IP subnetworks, IP ports, etc. [Cis96] [Cis97a] [Cis97c]

By building a central intelligent agent and a way to monitor bottlenecks for certain situations, and by using the enforcement capabilities already available in some routers, an effective bottleneck management system should be able to be constructed.

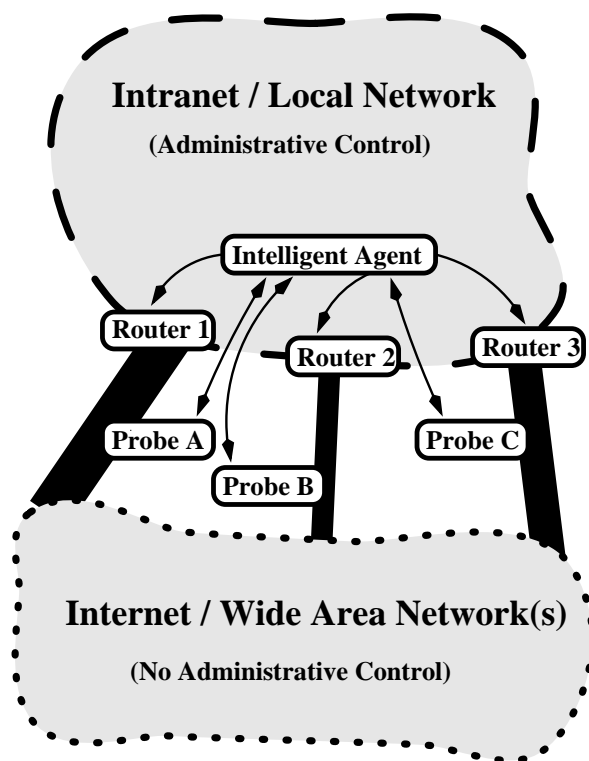


Figure 3.4 Distributed Multi-segment Bandwidth Management

A multi-homed site manages its aggregate extra-site bandwidth by registering its policies with an intelligent agent. The intelligent agent then tells each probe what situations to watch for. When a probe detects one of those situations, it tells the intelligent agent which takes appropriate action and tells appropriate router(s) what action they should take.

4. PROTOTYPE DESIGN

The idea of an integrated bandwidth management system may seem reasonable. This chapter describes a design for a prototype of an integrated bandwidth management system, and comments on some of the resources that such an implementation is likely to make use of.

4.1 Design Overview

Overall, this prototype design consists of the three pieces discussed earlier in this paper:

- The observation agent (the probe) watches network traffic for certain configured conditions and reports their occurrence to the intelligent agent.
- The intelligent agent reads a configuration file containing descriptions of the bottlenecks that the prototype will manage and the policies for management. The intelligent agent configures the observation agent(s) with the conditions that the intelligent agent wants to be informed of, makes enforcement decisions when it receives an alarm¹, and informs the enforcement agent(s) of any new enforcement policies.
- The enforcement agent either forwards or drops each packet so that the resulting traffic conforms with its enforcement policies.

¹When the observation agent detects a configured condition on the network, it sends an *alarm* to the intelligent agent.

4.2 Design Assumptions

There are two major sets of assumptions that this design makes: the first set of assumptions considers support that is available from some routers, and the second considers where on the internetwork the various pieces of this implementation must be placed.

4.2.1 Router Considerations

It is expected that the routers can already act as enforcement agents. This is not an unreasonable expectation, as that support already exists in some vendors' routers. For instance, with the `traffic-shape` [Cis97c] and `access-list` [Cis97b] in Cisco's IOS (Internetwork Operating System), a router's user can set and enforce limits to the rate that packets of certain characteristics are forwarded.

4.2.2 Device Placement

The observation agent must see all of the traffic that is going to and coming from the bottleneck that the observation agent is monitoring. Therefore, this design assumes that the probe can be placed in such a way that it can actually see all of the traffic that it is supposed to observe. The probe may be placed so that it can see more than the traffic that it is monitoring, however one probe must see at least the traffic that it is supposed to observe. (Figures 4.1 and 4.2).

This assumption does limit the sorts of networks that can be monitored. Specifically, if the managed bottleneck is located off of a non-binary ² router and the probe cannot be placed on the bottleneck side of the router, then this prototype design cannot manage that bottleneck (Figure 4.3).

²A binary router is a router that connects only two network segments. A non-binary router routes among three or more network segments.

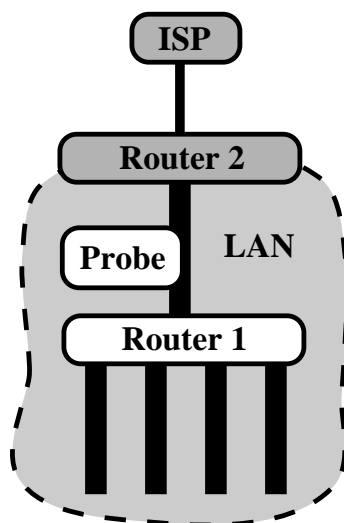


Figure 4.1 Acceptable Probe Placement 1

The probe is located on a segment between an organization's router and a router belonging to and managed by its ISP. This is a common situation when an organization has a large network. Its ISP does not manage the organization's internal network, but the organization does not maintain its own internet connectivity.

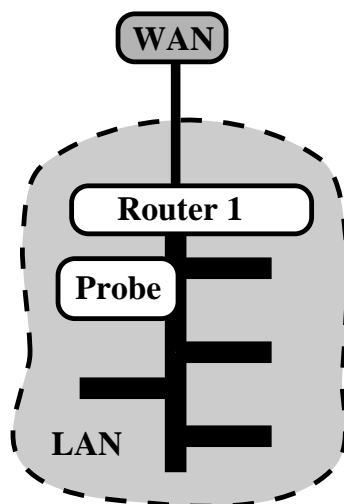


Figure 4.2 Acceptable Probe Placement 2.

The probe is located on a network segment where it can see all of the traffic that goes through the bottleneck to the wide area network.

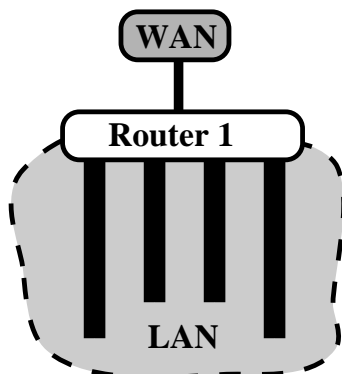


Figure 4.3 Probe Cannot Observe Bottleneck

This is an example of a bottleneck that this design is unable to manage. Since a probe cannot be placed on the bottleneck side of the the router, and since there is no other point at which a probe could observe all of the traffic going into and coming out of the bottleneck, this bottleneck cannot be managed by this design. If probes were deployed on each of the four routed LAN segments, then each probe could report only on the portion of the traffic that it “sees”. The probes have no way of adding the traffic that they “see” together.

The enforcement agent must be placed such that all of the traffic that is bound for the bottleneck and all of the traffic that is coming from the bottleneck passes through the enforcement agent. For this prototype it is assumed that the enforcement agent is a router.

Finally, there are no assumptions about where the intelligent agent must be placed, provided that the intelligent agent is in communication with the other agents.

4.3 Design Considerations and Solutions

This section describes several of the most important design considerations for a bottleneck management tool and how they were addressed in the design for the prototype.

4.3.1 Deployment Requirements

There are three requirements for the deployment of this design:

- **Unilaterally Deployable**

The deployment of this design should depend only on the approval of the owners of the network that it is to be deployed on. Deploying this tool should not require the help or support or approval of any other service or entity. This approach does not depend upon any other organization or entity for support.

- **Safely Deployable**

Routers were not made to be repeatedly reconfigured while operating. Reconfiguring for many routers means not accepting or forwarding packets for several seconds. Furthermore, many of the advantages to bottleneck management can be had by simply alerting human administrators to troublesome situations. Therefore, there is no requirement that the intelligent agent modifies the router configuration. Instead, it may simply alert human administrators. The operators of an implementation should be able to turn off the portions of the implementation that they consider “unsafe”.

- **Incrementally Deployable**

This is an incrementally deployable approach to bottleneck management because it can start by managing a single bottleneck, then a second could be added, then a third, etc. It can manage as many bottlenecks or as few bottlenecks as the administrators wish. This prototype is designed such that it is easy to add management to bottlenecks and remove management from bottlenecks.

4.3.2 Scalable

Many parts of this design are scalable. The three major pieces – the intelligent agent, observation agent, and enforcement agent – distribute the load of this application well. If an intelligent agent or an observation become overwhelmed with work, another one can be easily added. The least scalable portion of this design is the enforcement agent on the router. The router requires processor time to determine if a packet should be allowed or disallowed based on the rules that the intelligent agent gives it [Cis97a]. Care should be taken to ensure that the routers do not get so busy that throughput suffers.

4.3.3 Centralized and Simplified

Part of the reason that the tools that are already available to manage bandwidth are not used may be that they are difficult to maintain and set up. By supplying a single point from which an administrator can specify how scarce bandwidth is to be used, the administrator is much more likely to manage bottlenecks that need to be managed.

While the intelligent agent effectively “centralizes” the design, it is the configuration language that is responsible for making this bandwidth bottleneck management tool “simplified.”

4.4 Distribution of Tasks

This section describes in detail the duties of each piece in this bandwidth management tool.

4.4.1 Intelligent Agent

The intelligent agent has four main responsibilities: it is responsible for the configuration information, the initialization of the observation agents, the receipt of alarms

from the observation agents and deciding what action to take, and configuring the enforcement agent portion of the router (to enforce the action that the intelligent agent decided to take).

The responsibilities of the intelligent agent include configuring the management system. The intelligent agent must read a configuration file which describes the location and size of each bottleneck that it is to manage, the address of the observation agent that can report back any alarms, and enough information about the specific router (which is acting as an enforcement agent) to be able to configure it to enforce the action that the intelligent agent decides to take.

The intelligent agent must contact each observation agent and pass each filter³ to every observation agent.

The intelligent agent waits for alarms from the observation agents. When the intelligent agent receives a filter, the intelligent agent checks its bottleneck management policy to determine if an action is required.

If an action is required, the intelligent agent translates that action into a form that the router (or enforcement agent) understands, and reconfigures the router so that it can enforce the intelligent agent's decision.

4.4.2 Probe / Observation Agent

The observation agent's three duties are to accept filters from a variety of intelligent agents, to determine when an alarm should go off, and when an alarm goes off, to inform the intelligent agent that set the filter of the alarm.

4.4.3 Enforcement Agent / Router

The enforcement agent is a router, and its main duty is to route packets. Many routers can also act as enforcement agents, and this design simply makes use of some of the abilities that some routers have to shape traffic.

³A **filter** is a set of parameters that, when true, indicate an alarm should go off.

5. IMPLEMENTATION

The prototype of this integrated bottleneck management system was implemented in C on a number of Sun SPARCstation 4s, 5s, and 20s running Solaris 2.5 and 2.5.1 (SunOS 5.5 and 5.5.1). This chapter describes the major pieces of the prototype, how they were implemented, and what their major limitations are.

5.1 The Observation Agent `probe`

The observation agent consists of two heavyweight processes [SG94] joined by shared memory [SG94] [Hay95]. One process is responsible for receiving filters and putting them into shared memory. The other process applies every filter to every incoming packet.

`Probe` is based on `pcapture` [MLJ96a] from Lawrence Berkeley Laboratories. Like `pcapture`, `probe` accepts a number of parameters¹:

- c Keep the last *count* packets. The default is 500. Packet sizes are added to the bandwidth counter for appropriate filters when a packet enters the probe, and subtracted from those appropriate filter bandwidth counters after *count* more packets are received. This option should be set so that it represents a reasonably long period of time.

¹Most of the options for `probe` are identical to the options for `pcapture`. Furthermore, the functions of the options are very closely related. The option descriptions for `probe` in this work are taken from the options for `pcapture` in `pcapture`'s man page, but were altered to properly describe `probe`.

- i Listen on *interface*. If unspecified, probe searches the system interface list for the lowest numbered, configured up² interface (excluding loopback). Ties are broken by choosing the earliest match.
- O Do not run the packet-matching code *optimizer*. This is useful only if you suspect a bug in the optimizer.
- p Don't put the interface into *promiscuous* mode. Note that the interface might be in promiscuous mode for some other reason; hence, '-p' cannot be used as an abbreviation for 'ether host local-hw-addr or ether broadcast'.
- s *Snarf* *snaplen* bytes of data from each packet rather than the default of 68 (with SunOS's NIT, the minimum is actually 96).
- w *Write* the raw packets to file. Standard output is used if file is "-". The file can later be processed with the -r option of `tcpdump(1)` [MLJ96b]. The default for the -w option is "/dev/null".
- P Listen for new filters on this *port*.

expression Specifies which packets will be examined and applied to the filters. If no expression is given, all packets on the net will be examined and applied to the filters. This feature is most useful when a probe can see more than just the traffic which is bound for or exiting the bottleneck that this instance of probe is watching. For more information on expressions, see `tcpdump` [MLJ96b].

5.1.1 Receiving Filters

The first of the two processes that make up probe is the process that receives new filters from the intelligent agent. This process simply accepts TCP connections, finds a free place in the filter database, and copies the received filter into the database.

²An "up" interface is one that is currently operating.

The database is in shared memory so that the process that applies the filter can apply incoming packets against the new filter.

When a new filter is installed, this process sets a flag indicating that the filter is valid but the sections of the filter maintained by `probe` are not initialized.

5.1.2 Filtering Packets

The packet filtering process is a combination of the structures that this process maintains and the actions that it takes to maintain them. Both of those are described in this subsection.

5.1.2.1 Data Structures

The filtering process' main data structures are its circular buffer and the filter data structure. The circular buffer keeps the last `count` packets seen on the wire, and the filter data structure holds all the information that quantifies how `probe` determines which packets match each filter and how many octets of packets in the circular buffer have matched each filter.

The circular packet buffer simply contains the packet itself and the `libpcap` [MLJ96c] packet header. Every packet, along with its `pcap` header, that is examined is placed in this buffer, and is overwritten after every position in the circular buffer is filled. If there are `n` positions in the circular buffer, then this structure holds the last `n` packets that were examined.

The filter structure contains all of the parameters that the probe can use to describe a packet, currently: trigger threshold, Ethernet source address, Ethernet destination address, IP source address and IP source mask, IP destination address and IP destination mask, IP protocol, IP source port, and IP destination port. The filter structure also contains all of the parameters that describe the current state of that filter, namely: the number of octets of packets in the circular buffer matching this filter, the sequential number of the packet when this filter was first applied, a flag

that indicates the state of the filter (active, received but not yet active, or inactive), the alarm state, and how much bandwidth in bits per second the packets that match this filter represent. Finally, the filter structure contains the IP address and port of the filter setter. (Appendix A on page 38)

5.1.2.2 Actions Taken to Filter Packets

Most of the work of `probe` is in the process that applies each incoming packet to each filter. This implementation uses the Berkeley Packet Filter (BPF) [MJ91] and the `pcap` library [MLJ96c] to select the packets which it will examine, and only the packets examined will be processed by the filter mechanism described here. When a new packet is received, it is “checked in.” If the new packet will replace an old packet in the circular packet buffer, the old packet is “checked out.” The new packet is then placed into the circular packet buffer. Finally, the program checks every filter to see if its conditions are satisfied (Figure 5.1).

The “check in,” the “check out,” and the check for new filters are all done in the same loop. For every filter, if it is new then the packet number is recorded in the filter structure and the filter state in the filter structure is updated. `Probe` then applies the filter to the new packet. If the filter matches the packet, then the size of the packet is added to the total size of the matching packets in the filter structure. This is the “check in.” If a packet in the circular buffer will be overwritten by the new packet, that packet is “checked out.” To do this, if the filter matches the packet in the circular buffer, then the size of saved packet is subtracted from the size of the matching packets in the filter structure. The “check in” and “check out” processes are summarized by the following equation:

$$NewSize = OldSize + PacketOnWire - OverwrittenPacket$$

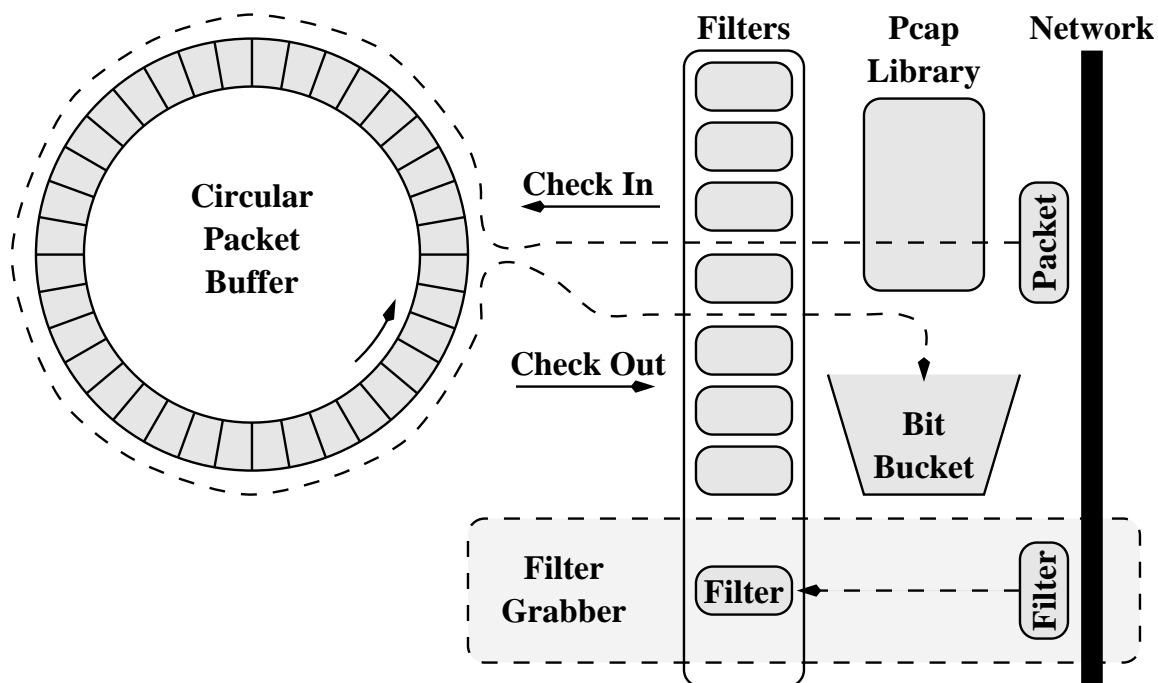


Figure 5.1 How probe Works

This figure shows the path a packet takes through **probe**, along with the mechanism **probe** uses to acquire new filters from the intelligent agent.

After the “check in” and “check out,” **probe** next saves the captured packet in the circular packet buffer, where it will stay until the time comes for it to be “checked out.”

Finally, **probe** goes back through every filter to determine if a filter should change state. To do that, it first calculates the amount of clock time represented by all of the packets in the circular packet buffer, uses that to calculate the amount of bandwidth that is represented by all of the packets in the buffer that match that filter, and if a filter changes state (from active to inactive or from inactive to active), **probe** informs the intelligent agent that set that filter.

5.2 The Intelligent Agent `intagent`

This section describes the major components of the prototype intelligent agent, `intagent`. `Intagent` accepts no parameters, but expects a configuration file on `stdin`.

5.2.1 The Language

The configuration language of `intagent` is two-tiered. The first tier describes a bottleneck that `intagent` is to manage, which includes the address of the enforcement agent, the interface of the enforcement agent, the bandwidth available through the bottleneck and the address of the probe. The second tier is embedded within the first, and contains the filter, which includes parameters which describe packets and the action that `intagent` is to take. An example follows, and the grammar can be found in Appendix B on page 42.

This example of an `intagent` configuration file declares two segments that `intagent` is going to manage. The first segment's enforcement agent is the router at `132.235.244.254`. The segment being managed is off of that router's `ethernet1` interface, and `intagent` can send filters to the machine `hudson.cns.ohiou.edu` on port `1234`. Furthermore, `intagent` will send three filters to `probe` on `hudson`. The first filter matches traffic to port `9` of any machine on the `132.235.1` network if its magnitude exceeds `500000` bits per second. The second filter matches any traffic on port `9` from any machine in the `132.235.3` network from the `132.235.244` network if its magnitude exceeds `100000` bits per second. Finally, the third filter matches any UDP traffic if its magnitude exceeds `100000` bits per second. The second segment's enforcement agent is the router at `132.235.3.254`. The segment being managed is off of that router's `ethernet0` interface, and `intagent` can send filters to the machine `jarok.cs.ohiou.edu` on port `2345`. Furthermore, `intagent` will send exactly one filter to `jarok`'s `probe`, and that filter will be triggered if UDP traffic exceeds `50000` bits per second.

```

{
# This starts the declaration of a segment.
Router 132.235.244.254
Interface ethernet1
Bandwidth 1000000
Probe hudson.cns.ohiou.edu 1234
  {
  # If busy, limit traffic from port 9 on the
  # 132.235.1 net to 500000 bits/sec.
  Limit 500000
  DSTaddr 132.235.1.0 255.255.255.0
  Port 9
  }
  {
  # Tell hstenzel if traffic on port 9 between the 132.235.3
  # and 132.235.244 networks exceeds 100000 bits/sec.
  Notify 100000 hstenzel@ohiou.edu "Help! IRG traffic!"
  SRCaddr 132.235.3.0 255.255.255.0
  DSTaddr 132.235.244.0 255.255.255.0
  Port 9
  }
  {
  # Don't let UDP traffic exceed 100000 bits/sec.
  Cap 100000
  Protocol UDP
  }
}
{
# This starts the declaration of another segment.
Router 132.235.3.254
Interface ethernet0
Bandwidth 1000000
Probe jarok.cs.ohiou.edu 2345
  {
  # If busy, limit UDP traffic to 50000 bits/sec.
  Limit 50000
  Protocol UDP
  }
}

```

Figure 5.2 Example Configuration File for intagent

5.2.2 Dealing with `probe`

`Intagent` passes filters to `probe` and `probe` passes filters back to `intagent` when they change from active to inactive or from inactive to active. Currently, `intagent` does not do anything with the filters that `probe` passes back.

5.2.3 Dealing with the Enforcement Agents

`Intagent` cannot configure routers to act as enforcement agents. Because requiring a router to reconfigure (and act as an enforcement agent) stops the router from performing its normal function. Furthermore, during the course of this work, access to a router capable of acting as an enforcement agent was lost.

5.2.4 Notable Limitations of `intagent`

In addition to the un-implemented portions of `intagent`, the language must be modified before `intagent` can be acceptably intelligent. Each filter should have a label, the actions portion of the filter structure should be removed, and a different block in the language should allow for the specification of a boolean algebra “rule” and the action which should accompany that rule.

5.3 Tests and Results

Two sets of tests were run to determine the effectiveness of `intagent` and `probe`. The first tests that were run tested the `probe`, and the second tested `intagent`.

5.3.1 Testing Probe

`Probe`'s effectiveness with various sized circular buffers was tested. On a 10 megabit per second ethernet network, `probe` was run with varying circular buffer sizes to determine the most useful sized circular buffers. The tests were carried out by giving `probe` nine filters: Each filter would match the TCP or UDP discard port.

The various filters would be set to alarm at 100000, 200000, 300000, 400000, 500000, 600000, 700000, 800000, and 900000 bits per second. Traffic was then generated on UDP discard port, and the reactions of `probe` recorded.

A circular buffer of 500 packets was too small. Alarms would be triggered then immediately reset, and the variety of packets on the network prevented even a very fast talker to, according to `probe`, use most of the network. The buffer represented at most 10 seconds of past packets.

A circular buffer size of 2000 packets performed reasonably well. Once `probe` detected the problem (in this case a flood of UDP packets to a discard port), it responded nearly immediately. After the “flood” passed, `probe` took between 10 and 30 seconds to turn the alarm off. The buffer represented 10 to 40 seconds of past packets (Table 5.1).

A circular buffer size of 10000 packets performed sluggishly. It was slow to alarm and slow to turn the alarm off. It represented between one and 3.5 minutes of past packets, and could take up to 8 minutes to turn an alarm off.

It is expected that network managers select a buffer size appropriate to their network. An appropriate buffer size should take into consideration factors such as the speed of the network and the amount and sort of traffic usually on the network. A buffer too large may not be able to address a problem, because by the time the alarm sounds, the problem has gone. On the other hand, a buffer too small may sound an alarm when a problem does not really exist.

5.3.2 Testing `intagent`

`Intagent`'s ability to manage multiple `probes` was tested by running three instances of `probe` and one instance of `intagent` all on different machines on the same network. `Intagent` was able to manage all three instances of `probe` to the extent that `intagent` is implemented.

Threshold (bits/sec)	Time Active (sec)	Buffer Size on / off (octets)	Time in Buffer on / off (sec)	Trigger on / off (bits/sec)
100000	46	494385 / 374698	39.4121 / 29.9878	100352 / 99960
200000	40	691832 / 628606	24.7735 / 25.1622	223410 / 199857
300000	36	830199 / 810878	22.1322 / 21.6841	300087 / 299160
400000	34	964480 / 957448	19.2556 / 19.1925	400706 / 399093
500000	32	1080809 / 1062184	17.2613 / 17.0229	500916 / 499179
600000	30	1168391 / 1146740	15.5406 / 15.3020	601466 / 599526
700000	28	1234603 / 1221860	14.0987 / 14.0326	700548 / 696586
800000	27	1293628 / 1280241	12.9296 / 12.8186	800413 / 798990
900000	26	1348647 / 1327074	11.9705 / 11.8080	901311 / 899099
Generated Traffic	15.33 sec	16384000 octets	n/a	816000 bits/sec

Table 5.1 Probe Performance With 2000 Packets in the Circular Buffer

6. CONCLUSIONS

A new type of bandwidth management, integrated bottleneck management, was introduced. This approach to bandwidth management has abilities that are unattainable with the other approaches to bandwidth management, approaches like the connection and segment orientated approaches.

6.1 Feasibility

Integrated bottleneck management is generally feasible. Most of the support required for the implementation proposed here already exists in some routers. However, there are networks segments that are not manageable under this method of bottleneck management. For those segments, additional network monitoring tools are necessary.

There is a need for explicit support from routers for a generalized method of bandwidth allocation that allows any duly authorized and verified agent to instruct the router to act as an enforcement agent. This mechanism should not disrupt the normal operation of the router and should be as accessible as a protocol, not just as a command-line configuration option is available now.

Finally, there is a need for better network monitoring tools. SNMP [Wal95] [Wal93] [Wal91] was not able to supply enough information to act as a full-featured observation agent.

6.2 Value

There is no other system that allows a site to manage its bandwidth, and there is a real need for such a system. Through this work, we conclude that a suitably

advanced and flexible mechanism, such as integrated bottleneck management, would greatly enhance the control that network managers have over their internetworks.

6.3 Future Work

Possibilities for future work include:

- Continue work on `intagent` and `probe`. These tools could be developed into a production network monitoring tool.
- Work on a bandwidth allocation protocol for routers. Such a protocol would allow any authorized bandwidth management agent to implement bandwidth management policies on the router supporting this protocol.
- Work on additional network monitoring tools. SNMP was unable to provide the support required for the bandwidth management tool described in this work.
- Examine other ways of managing bandwidth in packet-switched networks. Very little work has been done in this area.

BIBLIOGRAPHY

BIBLIOGRAPHY

- [Cis96] Cisco Systems. *Cisco IOS Software Features for Differentiated Class of Service for Internetworks*, October 1996.
- [Cis97a] Cisco Systems. *Advanced Queuing Techniques in the Cisco IOS*, April 1997.
- [Cis97b] Cisco Systems. *IP Commands*, May 1997.
- [Cis97c] Cisco Systems. *System Management Commands*, May 1997.
- [Com88] Douglas E. Comer. *Internetworking with TCP/IP Volume I, Principles, Protocols, and Architecture*. Prentice-Hall, Englewood Cliffs, New Jersey, 1988.
- [Hay95] Steven Hayes. An Example of Shared Memory, August 1995. <http://www.cs.rmit.edu.au/~steveh/tns/solaris/node68.html>.
- [IBM94] IBM. *Asynchronous Transfer Mode (Broadband ISDN) Technical Overview*, June 1994.
- [Lyn96] Lynx Technologies. *Frame Relay – Service Description*, November 1996.
- [MJ91] Steven McCanne and Van Jacobson. Berkeley Packet Filter, July 1991. <ftp://gatekeeper.dec.com:pub/DEC/bpfext.tar.Z>.
- [MLJ96a] Steve McCanne, Craig Leres, and Van Jacobson. *Pcapture - Capture the Last Few Packets*, July 1996. <ftp://ftp.ee.lbl.gov/pcapture.tar.Z>.
- [MLJ96b] Steve McCanne, Craig Leres, and Van Jacobson. *Tcpdump - Dump Traffic on a Network*, August 1996. <ftp://ftp.ee.lbl.gov/tcpdump.tar.Z>.
- [MLJ96c] Steven McCanne, Craig Leres, and Van Jacobson. *Pcap - Packet Capture Library*, August 1996. <ftp://ftp.ee.lbl.gov/libpcap.tar.Z>.
- [Pos81] Jon Postel. Transmission Control Protocol, September 1981. RFC 793.

- [Pro96] Progressive Networks. *Delivering time-based information over the Internet: HTTP versus RealAudio Client-Server Streaming*, October 1996.
- [Sat96] Michael Sattler. CU-SeeMe, June 1996. <http://www.indstate.edu/msattler/sci-tech/comp/CU-SeeMe/index.html>.
- [SG94] Abraham Silberschatz and Peter Galvin. *Operating Systems Concepts (Fourth Edition)*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.
- [Soc91] T. Socolofsky. A TCP/IP Tutorial, January 1991. RFC 1180.
- [Wal91] S. Waldbusser. Remote Network Monitoring Management Information Base, November 1991. RFC 1271.
- [Wal93] S. Waldbusser. Token Ring Extensions to the Remote Network Monitoring MIB, September 1993. RFC 1513.
- [Wal95] S. Waldbusser. Remote Network Monitoring Management Information Base, February 1995. RFC 1757.
- [ZDE93] Lixia Zhang, Stephen Deering, and Deborah Estrin. RSVP: A New Resource ReSerVation Protocol. Novel Design Features Lead to an Internet Protocol that is Flexible and Scalable. *IEEE Network*, 7(5):8, September 1993.

APPENDIX

A. FILTER.H

Appendix A contains `filter.h`, the C include file that contains the structures and definitions that describe filters and segments throughout `intagent` and `probe`.

```
/*
 * filter.h
 *
 * The data structures used.
 *
 * Harley A. Stenzel
 *
 */

#define SHMFILTERKEY    75

#define FILTER_GRABBER_RETURN_INUSE    -1
#define FILTER_GRABBER_RETURN_ERROR    -2

#include <sys/param.h>
#include <sys/time.h>
#include <sys/socket.h>

#include <net/if.h>

#include <netinet/in.h>
#include <netinet/if_ether.h>
#include <netinet/in_sysm.h>
#include <netinet/ip.h>
```

```

#include <netinet/ip_var.h>

struct filter {
    /* parameters that describe what the filter watches for */
    u_long filter_type;          /* which fields are valid */
#define FILTER_TYPE_NOFILTER    0
#define FILTER_TYPE_ETHER_SRC  1 /* or together defined fields */
#define FILTER_TYPE_ETHER_DST  2
#define FILTER_TYPE_IP_SRC     4
#define FILTER_TYPE_IP_DST     8
#define FILTER_TYPE_IP_PROTOCOL 16
#define FILTER_TYPE_IP_SRCPORT 32
#define FILTER_TYPE_IP_DSTPORT 64
    u_long threshold;          /* bits/sec activated on */
    struct ether_addr ether_src_addr; /* Ethernet source address */
    struct ether_addr ether_dst_addr; /* Ethernet destination addr */
    struct in_addr ip_src_addr;    /* IP source address */
    struct in_addr ip_src_mask;    /* IP source mask */
    struct in_addr ip_dst_addr;    /* IP destination address */
    struct in_addr ip_dst_mask;    /* IP destination mask */
    u_long ip_protocol;          /* protocol filter flags */
#define FILTER_PROTOCOL_TCP    2
#define FILTER_PROTOCOL_ICMP   4
#define FILTER_PROTOCOL_IP     7 /* Cisco:IP = UDP & TCP & ICMP */
    u_long ip_src_port;          /* IP source port */
    u_long ip_dst_port;          /* IP destination port */

    /* parameters that describe to whom the filter reports */
    u_long label;                /* label to identify a filter */
    struct in_addr owner_addr;    /* Who should I notify? */
    u_long owner_port;           /* Which port? */

```

```

    /* variables modified by the filter processing mechanism */
    u_long filter_bsize;          /* octets of filter in buffer */
    u_long start_packet;         /* pkt num when filter active */
    u_long validflag;           /* the state of the filter */
#define FILTER_VALID_INVALID    0 /* the filter is free for use */
#define FILTER_VALID_EXISTS    1 /* created but not activated */
#define FILTER_VALID_ACTIVE    2 /* activated */
    u_long alarm_state;         /* Is the filter alarming? */
#define FILTER_ALARM_INACTIVE  0
#define FILTER_ALARM_ACTIVE    1
    u_long bandwidth;          /* the amt of bandwidth used */

    /* variables that are maintained by the intelligent agent */
    u_long action;              /* what the i.a. should do */
#define FILTER_ACTION_NONE      0
#define FILTER_ACTION_NOTIFY    1
#define FILTER_ACTION_LIMIT     2
#define FILTER_ACTION_CAP       3
#define FILTER_EMAIL_LEN       64
    char email[FILTER_EMAIL_LEN]; /* the email address to notify */
#define FILTER_MSG_LEN         128
    char msg[FILTER_MSG_LEN];    /* the message for a notify */
};

struct ia_segment_def {
    char *router;                /* address or name of router */
    char *interface;            /* Interface of the router */
    char *probe;                /* the address of the probe */
    u_long port;                /* the port of the probe */
    u_long bottleneck;         /* bits/sec through bottleneck */
    u_long numfilters;         /* num filters actually used */
#define MAX_NUM_IA_FILTERS    20

```

```
    struct filter filter[MAX_NUM_IA_FILTERS];    /* Watch for this */
};

struct ia_segments {
    u_long numsegments;           /* how many segments are used */
#define MAX_NUM_IA_SEGMENTS      5
    struct ia_segment_def segment[MAX_NUM_IA_SEGMENTS]; /* segments */
};

struct probe_filters {
#define MAX_NUM_PROBE_FILTERS    20
    struct filter filter[MAX_NUM_PROBE_FILTERS]; /* array of filters */
    u_char goaway;
};
```

B. GRAMMAR

Appendix B contains the grammar for the configuration file for `intagent`. Something surrounded by “‘ ’” is literal, something surrounded by “<>” is a value, and everything else is a label for the grammar.

```
configuration:  segments
```

```
segments:
```

```
    segment
    | segment segments
```

```
segment:
```

```
    ‘{’
    router
    interface
    bandwidth
    probe
    rules
    ‘}’
```

```
router:
```

```
    ‘ROUTER’
    <ipaddress>
```

```
interface:
```

```
    ‘INTERFACE’
```

```

    <string interfacename>

bandwidth:
    'BANDWIDTH'
    <integer bandwidth>

probe:
    'PROBE'
    host
    <integer port>

host:
    <ipaddress>

rules:
    rule
    | rule rules

rule:
    '{' action '}'

action:
    'NOTIFY' <integer threshold> <string address> message filters
    | 'LIMIT' <integer threshold> filters
    | 'CAP' <integer threshold> filters

message:
    '""' words '""'

words:
    <string>
    | words <string>

```

filters:

```
filter
| filter filters
```

filter:

```
‘‘ADDR’’ addrpair
| ‘‘SRCADDR’’ addrpair
| ‘‘DSTADDR’’ addrpair
| ‘‘PROTOCOL’’ protocol
| ‘‘PORT’’ <integer port>
| ‘‘SRCPORT’’ <integer port>
| ‘‘DSTPORT’’ <integer port>
```

addrpair:

```
<ipaddress network> <ipaddress mask>
```

protocol:

```
‘‘IP’’
| ‘‘UDP’’
| ‘‘TCP’’
| ‘‘ICMP’’
```