ON PROXY SERVER BASED MULTIPATH CONNECTION

by

YU CAI

B.S., Zhong-Shan University, 1996

A dissertation submitted to the Graduate Faculty of the

University of Colorado at Colorado Springs

in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

Department of Computer Science

2005

This dissertation for the degree of Doctor of Philosophy by

Yu Cai

Has been approved for the

Department of Computer Science

by

Dr. Edward. Chow, Chair

Dr. Terry Boult

Dr. Charlie Shub

Dr. Xiaobo Zhou

Dr. Rodger Ziemer

Date

Yu Cai (Ph.D., Computer Science)

On Proxy Server based Multipath Connection

Dissertation directed by Professor Edward Chow

A multipath connection provides multiple paths among network hosts. The traffic from a source can be spread over multiple paths and transmitted in parallel through the network. The receiver collects the incoming network packets, re-assembles them, and delivers them to the upper-level end users. Multipath connections offer applications with the ability to improve network performance, security and reliability.

In this dissertation, techniques for supporting the proxy server based multipath connection (PSMC) are studied. First, the design and implementation of a proxy server based overlay network using a set of intermediate connection relay proxy servers is presented. Multiple indirect alternate paths can be set up via these proxy servers. The proxy server based overlay network is used in a Secure Collective Defense system (SCOLD) to defend against Distributed Denial of Services (DDoS) attacks. The Berkeley Internet Name Domain (BIND - v.9) package is enhanced to support indirect routing with IP Tunneling. The performance of the SCOLD system validates the capability of PSMC in enhancing the network security.

Second, the existing TCP/IP protocol is enhanced with a proxy server based multipath protocol (PSMP). On the sender side, the IP layer is enhanced to distribute packets across multiple paths. The TCP congestion window control is revised for higher throughput. On the receiver side, the TCP layer is enhanced with a double buffer to solve the persistent reordering problem. Detailed analysis of the PSMP is presented. A communication

channel is set up between the sender and the receiver for exchanging network traffic information. The enhancement supports both Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

Third, proxy server selection algorithms are developed for selecting a subset of proxy servers from a large set of available proxy servers with various object functions and constraints.

Forth, resource allocation schemes are proposed and implemented on the end server to provide proportional differentiated services. These schemes are based on the queueing theory and feedback control theory. By combining the multipath on the network with service differentiation at the end server, a comprehensive solution for various QoS and security related applications can be provided.

PSMC utilizes existing network protocols and infrastructure with some enhancements. This ensures the ease of its deployment with the current Internet in various network environments. Therefore, a large number of applications could benefit from utilizing PSMC. The research results and insight obtained from PSMC could have broader impact on the protocols and security in today's Internet.

Dedication

This thesis is dedicated to my parents Tong-zhi and Lang-feng,

and to my soul mate Shu-han.

Acknowledgements

I am grateful for the support of professors, friends and my family. I would not reach the completion of this long journey without them. They make this challenging Ph.D. experience also a memorable one.

Many people have shared their time and expertise to help me accomplish my goal. First, I would like to sincerely thank my advisor, Dr. C. Edward Chow, for his constant support and guidance. It was him who brought this exciting research topic to me and guided me through the whole research. His worldwide around-the-clock instant responses to my countless inquiries have been invaluable and motivational. I have been privileged to have him as my advisor.

A sincere gratitude goes to Dr. Xiaobo Zhou. I have been receiving tremendous help from his professional and personal advices. He has patiently taught me how to analyze problems and write technical papers, which is very important for a researcher.

Many thanks to Dr. Terry Boult, who gives me valuable advice in my research. I am so impressed by his brilliant mind and wealth of knowledge. Thanks to Dr. Charles Shub, whose lectures have inspired me on my current research. Thanks to Dr. Rodger Ziemer for his encouragement and reassurance at times of frustration.

I wish to pay special tributes to a much-cherished friend, Ganesh Kumar Godavari. I am in deep appreciation for his willingness to share his knowledge, discuss with me on my research and stay with me in school for many sleepless nights.

I am grateful to Dave Lohmann for troubleshooting my network and providing priceless support during my research. Also deserve much credit are Dr. Augusteijn, Mrs. Rhea, the staff in the Computer Science department and my many friends. I would like to recognize the Network Information and Space Security Center (NISSC) and University of Colorado at Colorado Springs for their partial financial support to this work.

Finally, I need to acknowledge three very important persons in my life, my mother – Lanfeng, my father – Tongzhi and my soul mate – Shuhan. Without their love, support, friendship, and faith, I will not be here today.

CONTENTS

TABLES

FIGURES

CHAPTER I

INTRODUCTION

Overview

The key challenge in today’s Internet is to improve network performance, security, and reliability for heterogeneous Internet participants. The current network connections are mostly over a single path. This single path connection model is simple and easy to implement. The tremendous success of today’s Internet is a credit to the original design. However, the single path connection is vulnerable to potential attacks, link breakage, or even traffic congestion. It may also under-utilize network resources and suffer from performance problems. Therefore, it does not always provide a good and reliable network connection.

Due to the increasing demands from the Internet on network performance, security, and reliability, the Internet is undergoing a number of significant changes. Various Internet enhancements and services have been suggested [AKAM, DSEC, RON01, SSav99, WAdj99, CCas02, MZha04, JChen98]. Multipath connections are one of them.

A multipath connection provides multiple paths among network hosts. The traffic from a source is spread over multiple paths and transmitted in parallel through the network (Figure 1.1). The receiver collects the incoming network packets, re-assembles them, and delivers them to the upper-level end users. A multipath connection makes better use of network resources by aggregating the available bandwidth on multiple paths.

Therefore, a multipath connection can significantly improve the network performance. By providing redundant paths or alternate paths, a multipath connection has better ability to cope with network congestion, link breakage, outrage, and potential attacks, thus improve network security and reliability.

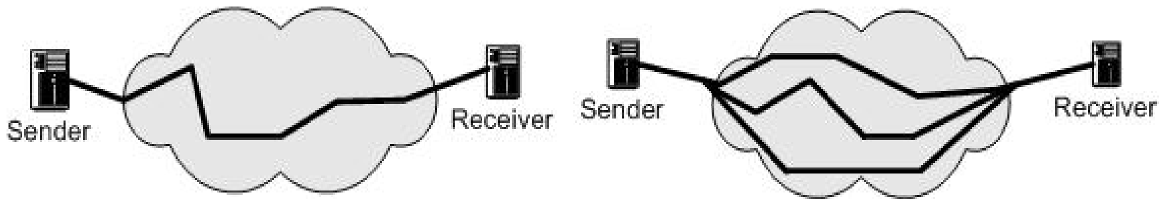


Figure 1.1: Single path connection vs. multipath connection

The IBM Systems Network Architecture (SNA) network in 1974 [SNA79] is probably the first attempt to provide multiple path connections among network nodes on wide area networks. N. F. Maxemchuk studied how to disperse the traffic over multiple paths in 1975. He called it “dispersity routing” [NMax75]. Since then, the idea of multipath connection has been studied in various settings. One example of multipath connection is link aggregation [LAgg], which is a data link layer protocol. In the IP layer, multipath connection has been studied extensively in the name of multipath routing. Various table-driving multipath routing algorithms (link state or distance vector) [SVJG01, SLMG00, ICRR99, SMJG96, NTBB99, WZJG98, SLMG00, ANSD99] and source routing algorithms [DJDM96, LZZZ02] were proposed. On the TCP layer, there have been works like [MZha04, HHsi02]. For more details, please refer to Chapter 2.

In this dissertation, we design and implement a novel multipath connection mechanism called the Proxy Server-based Multipath Connection (PSMC).

Figure 1.2 is a diagram that illustrates a PSMC network. There are three basic components in a PSMC network. The **multipath sender**, or distributor, is responsible for efficiently and adaptively distributing packets over the selected multiple paths. Some of the packets will go through the normal direct route; other packets will go through the alternate indirect routes via the proxy servers. The intermediate connection-relay **proxy servers**, or forwarders, examine the incoming packets and forward them to the destination through the selected paths. The **multipath receiver**, or collector, collects the packets arrived from multiple paths, reassembles them in order, and delivers them to the end user.

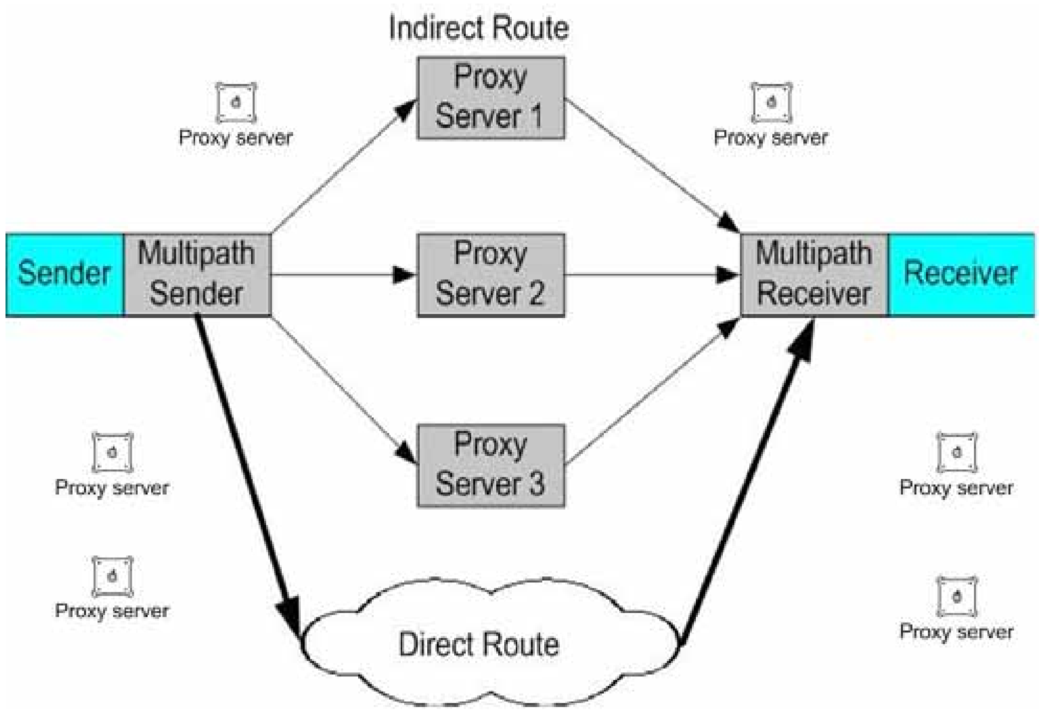


Figure 1.2: Proxy server-based multipath connection (PSMC)

The key features of PSMC are summarized as follows.

a) A proxy server-based overlay network is designed and implemented by using a set of intermediate connection relay proxy servers. Multiple indirect or alternate paths can be set up via these proxy servers.

b) A proxy server-based multipath protocol is designed and implemented by enhancing the existing TCP/IP protocol to effectively distribute, transport, and reassemble network packets over the multiple indirect paths between two end hosts.

c) Proxy server selection algorithms are designed and implemented to select a subset of proxy servers from a large set of available proxy servers with various object functions and constraints.

d) Resource allocation schemes are proposed and implemented on the end server and server cluster to provide proportional service differentiation. These schemes are based on queueing theory and feedback control theory. Combining the multipath on network with service differentiation on the end server, a comprehensive solution for various QoS and security related applications can be provided.

For convenience, from now on, we refer our approach of a proxy server-based multipath connection as "PSMC". We use the term "direct route" to refer to the network route which a packet normally takes when it travels through the network. The term "indirect route" is used to refer to the network route which utilizes the connection relay proxy server. The term "proxy server" is used specifically for the connection relay proxy servers in a PSMC network unless otherwise specified. We mix the usage of “route” and “path”.

In addition to the general benefits provided by a multipath connection, PSMC has the following unique advantages:

a) **Ease of Deployment**: PSMC utilizes and enhances the existing TCP/IP protocol and network infrastructure to distribute, transport, and reassemble packets. Unlike some multipath connection approaches like link aggregation and multipath routing, which require significant changes on network infrastructure, PSMC is built on an overlay network and only requires some feasible changes on network software and protocols on the end systems and the proxy servers. This ensures the ease of deployment with the current Internet. Therefore, PSMC can be more conveniently and adaptively deployed in various network environments. PSMC also has good scalability with regard to network size and number of proxy servers.

b) **Flexibility and usability**: PSMC is transparent to the application level end users. The end user can easily set up, manage, and maintain the multipath connection. PSMC also gives the end users more control and flexibility on multipath connection. A large number of applications in various categories could benefit from utilizing PSMC. For example, it can be used to defend against Distributed Denial of Service (DDoS) attacks with intrusion tolerance. Particularly, it can be used to defend a Domain Name System (DNS) Root Server against DDoS attacks. PSMC can also be utilized to provide an alternate or backup route and additional bandwidth based on operational requirements in an enterprise network. PSMC can be utilized to provide Quality of Service (QoS) for various applications.

Contributions

The contributions of this dissertation are summarized below.

Contribution 1

A proxy server based overlay network using a set of intermediate connection relay proxy servers is designed and implemented. Multiple indirect or alternate paths can be set up via these proxy servers.

The proxy server based overlay network is used in a Secure Collective Defense system (SCOLD) to defend against DDoS attacks. SCOLD provides alternate routes via a set of proxy servers and alternate gateways when the normal route is unavailable due to DDoS attacks. The BIND9 DNS server and its DNS update utilities are enhanced to support new DNS entries with indirect routing information. The indirect route is implemented by utilizing an IP tunnel. Protocol software for supporting the establishment of indirect routes based on the new DNS entries is developed for Linux systems.

Contribution 2

A proxy server based multipath protocol is designed and implemented by enhancing the existing TCP/IP protocol to effectively distribute, transport, and reassemble network packets over the multiple indirect paths between two end hosts.

We modify the Linux kernel to support the enhanced TCP/IP protocols. On the sender side, the IP layer is enhanced to distribute packets across multiple paths. The TCP congestion window control is also revised for higher throughput. On the receiver side, the TCP layer is enhanced with a double buffer to solve the TCP packet persistent reordering problem over multiple paths. A communication channel is set up between sender and receiver for exchanging network traffic information. The PSMC supports both TCP and UDP, which enables PSMC to support multimedia applications in today’s Internet.

Contribution 3

Proxy server selection algorithms are developed to select a subset of proxy servers from a large set of available proxy servers to meet various object functions and constraints.

Different sever selections may result in significantly different network performance. Therefore, server selection is a critical decision in a multipath system. When there are hundreds of proxy servers available, disjoint paths are more desirable because the route correlation can be reduced and network reliability and throughput can be improved. We have also developed heuristic algorithms to choose the best mirror sites for parallel download from multiple mirror sites.

Contribution 4

Resource allocation schemes on the end server and server cluster are designed and implemented to provide proportional differentiated services.

These schemes are based on queueing theory and feedback control theory. A process allocation approach on the Apache Web server is presented for proportional responsiveness differentiation.

Combining the multipath on the network with service differentiation on the end server, a comprehensive solution for various QoS and security related applications can be provided.

The rest of the dissertation is organized as follows. Chapter 2 presents the background and related work. Chapter 3 presents the idea of a proxy server-based overlay network (SCOLD). Chapter 4 presents the proxy server-based multipath protocol (PSMC). Chapter 5 presents the proxy server selection algorithm and its performance analysis. Chapter 6 studies the proportional service differentiation on end server and server cluster. Chapter 7 contains the conclusion and suggests future work.

CHAPTER II

RELATED WORK

This chapter surveys related work and background for the idea of multipath connections.

Multipath Connection

The technique of multipath connection appears under many different labels, like multiple path routing, alternate path routing, and traffic dispersion. Often the same label is used in the literature to refer to different things. We try to survey and clarify the different concepts of multipath connection in this section.

The IBM SNA network in 1974 [SNA79] is probably the first wide area network which provides multiple path connections between network nodes. However, in the SNA network, only one path is used at a time, and the purpose of multiple paths is to provide a fault-tolerance mechanism. Also, SNA multiple paths are predefined and pre-computed.

Maxemchuk [NMax75] in 1975 used channel sharing to provide multipath connections and reduce queuing delay in store-and-forward networks. He called the technique “dispersity routing”. This research was extended to virtual circuit networks and ATM networks to deal with busty traffic data, where both redundant and nonredundant dispersity routing techniques were described.

According to the Open System Interconnection (OSI) Network Reference Model [OSI], we try to differentiate multipath connections between the physical layer, data link layer, network layer, transport layer, and application layer. This is only a rough classification. Some approaches might be multiple layer implementations. Figure 2.1 is a diagram illustrated the classifications for multipath connections.

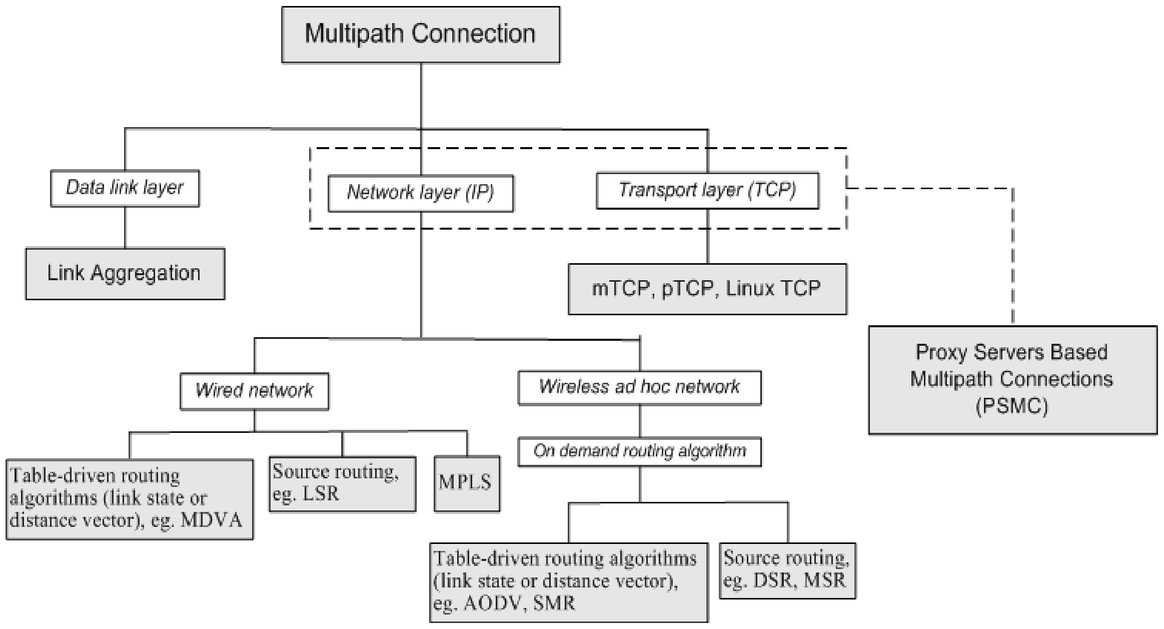

Multipath Connection
Data link layer
Network layer (IP)
Transport layer (TCP)
Link Aggregation
mTCP, pTCP, Linux TCP
Proxy Servers Based Multipath Connections (PSMC)
Wired network
Wireless ad hoc network
On demand routing algorithm
Table-driven routing algorithms (link state or distance vector), eg. MDVA
Source routing, eg. LSR
MPLS
Table-driven routing algorithms (link state or distance vector), eg. AODV, SMR
Source routing, eg. DSR, MSR

Figure 2.1: Diagram illustrating multipath connections.

Physical layer

Multipath connections in the physical layer are not always something that we want. For example, sometimes FM radio sounds noisy because of "*multipath interference*" [ERun]. Multipath interference happens when FM signals reflect from buildings in a city or other large obstructions. These reflections interfere with each other and the FM radio tries to demodulate the original signal as well as the reflection! Other usages of multipath connections in physical layer, like antenna arrays, are beyond the scope of this dissertation.

Data link layer

Multipath connections in the data link layer have been implemented as link aggregation or trunking, defined in IEEE 802.3ad [LAgg]. It is a method of combining multiple physical network links between two devices into a single logical link for increased bandwidth. The upper layer applications or protocols, such as a MAC client, can treat the link aggregation group as if it were a single link. Link aggregation requires special network hardware and software support. Therefore, it is only suited for high-end users. See Figure 2.2.

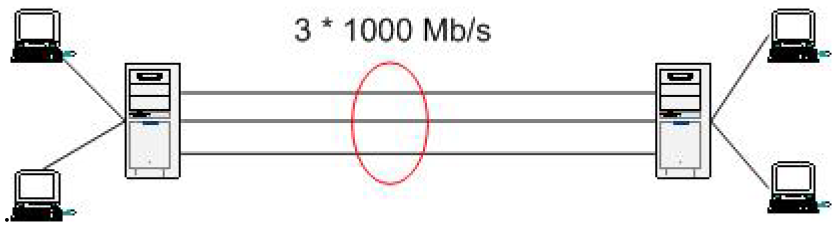


Figure 2.2: Two servers interconnected by link aggregation [LAgg]

Network layer

In the network layer, multipath connections have been studied extensively in the name of multipath routing. Various protocols have been designed for wired networks and wireless ad hoc networks.

a) Wired Networks

Based on the routing mechanism, we differentiate between table-driven algorithms (link state or distance vector) and source routing.

Table-Driven Algorithms

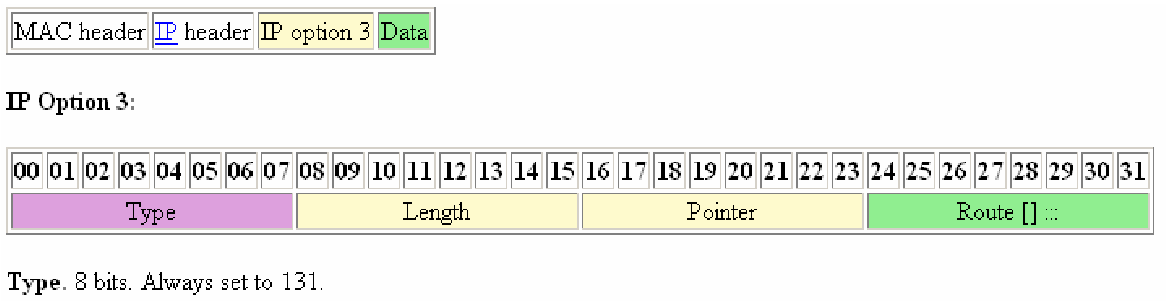
Vutukury et al. [SVJG01] proposed a multipath distance vector routing algorithm named Multipath Distance-Vector Algorithm (MDVA). It uses a set of loop-free invariants to prevent the count-to-infinity problem. The computed multipaths are loop-free at every instant.

Chen, in his Ph.D. dissertation [JChen98], proposed a complete multipath network model that includes the following three components: routing algorithms that compute multiple paths; a multipath forwarding method to ensure that data travel their specified paths; and an end-host protocol that effectively uses multiple paths.

Other works in similar areas include [ICRR99, SMJG96, ROVR93, DSRN91, NTBB99, WZJG98]. These protocols use table-driven algorithms (link state or distance vector) to compute multiple routes. These protocols require fundamental changes on Internet routers and routing protocols. Therefore, the usage and deployment of these algorithms and protocols are limited.

Source Routing

Source routing is a technique whereby the sender of a packet can specify the route that the packet should take when the packet travels through the network. In today's Internet, when a packet travels through the network, each router will examine the "destination IP address" and choose the next hop to forward the packet. In source routing, the sender makes some or all of these decisions. If the sender makes only some of these decisions, it is called loose source routing. Source routing could be used to implement multipath routing. But, because of the security concerns of source routing, most routers in today's Internet have disabled the source routing.



MAC header	IP header	IP option 3	Data

IP Option 3:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Type								Length								Pointer								Route [] :::							

Type. 8 bits. Always set to 131.

Figure 2.3: Datagram format for loose source routing

MultiProtocol Label Switching

Multiprotocol label switching (MPLS) provides a mechanism for engineering network traffic patterns that is independent of routing tables. MPLS assigns short labels to network packets that describe how to forward them through the network. MPLS is independent of any routing protocol.

In the traditional Level 3 forwarding paradigm, as a packet travels from one router to the next, an independent forwarding decision is made at each hop. The IP network layer header is analyzed, and the next hop is chosen based on this analysis and on the information in the routing table. In an MPLS environment, the analysis of the packet header is performed just once when a packet enters the MPLS cloud. The packet is then assigned to a stream, which is identified by a label, which is a short (20-bit) fixed-length value at the front of the packet. Labels are used as lookup indexes into the label forwarding table. For each label, this table stores forwarding information. Additional information can be associated with a label, such as class-of-service (CoS) values, that can be used to prioritize packet forwarding. MPLS could be used to set up multipath connections for traffic engineering and quality of service.

b) Wireless ad hoc network

Multipath routing in ad hoc wireless network is a topic gaining interest, and much work has recently been done in this field. An ad hoc wireless network is a collection of wireless mobile hosts forming an instant deployable network without the aid of any base station, other infrastructure or centralized administration. The most popular routing approach in ad hoc network is on-demand routing because of its effectiveness and efficiency. Routing protocols used in wired network, which periodically exchanging route messages to maintain route table, are not well suited for ad hoc network, due to the considerable overhead produced by route update and their slow convergence to topological changes. On-demand routing protocols build routes only when a node needs to send data packets to a destination. Each node operates as a specialized router, and routes are obtained on-demand with no reliance on periodic advertisements.
Based on the routing mechanism, we differentiate between Table-driven algorithms (link state or distance vector) and Source Routing.

Table-driven algorithms (link state or distance vector)

C. Perkins et al. [CPER99] proposed a novel algorithm for the operation of ad-hoc networks, named Ad-hoc On Demand Distance Vector Routing (AODV). The routing algorithm is quite suitable for a dynamic self-starting network, as required by users wishing to utilize ad-hoc networks.

Multipath routing protocols in ad hoc network proposed in [SLMG00], [ANSD99] are really backup route protocols, in the sense that even though these protocols build multiple paths on demand, but the traffic is not distributed into multiple paths. Only one route is primarily used and the secondary path is used when the primary route is broken.

S. Lee et al. [SLMG00-1] propose an on-demand multipath routing scheme for ad hoc wireless network, called Split Multipath Routing (SMR), that establishes and utilizes multiple routes of maximally disjoint paths. The proposed protocol uses a per-packet allocation scheme to distribute data packets into multiple paths of active sessions.

Source Routing

Dynamic Source Routing (DSR) proposed by D. Johnson et al. [DJDM96] is an enhanced source routing designed specially for wireless ad hoc network. The protocol is composed of two main mechanisms of “Route Discovery” and “Route Maintenance”, which together allow ad hoc nodes to discover and maintain routes to any destinations in the ad hoc network. This protocol allows multipath routing and allows sender to select the route(s) to use.

L. Wang et al. [LZZZ02] proposed a Multipath Source Routing (MSR) protocol for ad hoc wireless networks based on Dynamic Source Routing. MSR extends DSR’s route discovery and route maintenance mechanism to deal with multipath routing. The proposed scheme distributes load balance between multiple paths based on the measurement of RTT.

Transport layer

Linux has its own implementation of multipath connection [CSim]. For convenience, we refer to it as “Linux multipath connection”. It is a solution for using multiple ISP connections (multi-homing) at the same time. Linux kernel needs to be patched to support “Advance Router” and “Multiple Path Routing” options. The Linux kernel distributes packets between multiple network connections in TCP layer. The solution’s configuration is complicated, and it fails to provide fail-over mechanism in case of failure of a connection. Also, it requires the host machine to have multiple network interfaces with multiple ISP connections.

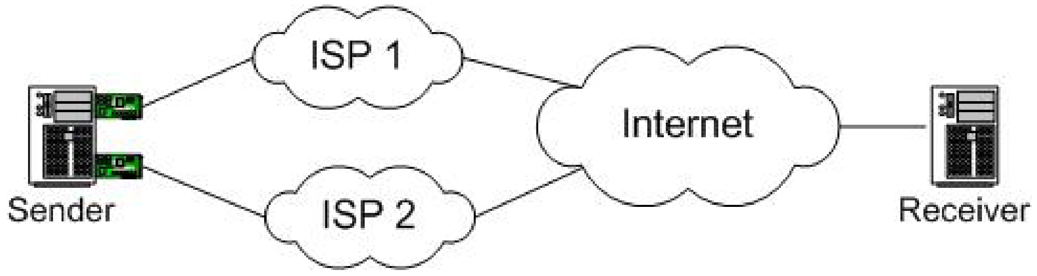


Figure 2.4: Linux multipath connection for multiple ISP connections

The closest multipath schemes on TCP layer to our PSMC work are mTCP [MZha04] and pTCP [HHsi02]. There are some chandelling issues in designing and implementing a TCP layer multipath solution. For more details, please refer to Chapter 4.

Both pTCP and mTCP are limited to TCP only, while PSMC supports TCP as well as UDP. Another major difference is that PSMC can be installed on one end-host (one-way multipath) or on two end-hosts (two-way multipath). In the first case, only the data packets from sender are spread out over multiple paths, the return ACK packets from receiver still go through the main direct path. In the second case, both the forwarding packets and the return packets are sent through multiple paths. pTCP is designed to support only one-way multipath.

Packet striping can occur on a different layer. The application layer [THac02, HSiv00] and data link layer [HAdi96, I802] implementations suffer from the inability to accurately profile the available bandwidth on individual paths. The TCP layer implementations like mTCP and pTCP use a different striping scheme by monitoring and keeping track of the outstanding packets on each path, which may impose operational overhead and a complicated mechanism.

Previous works for TCP persistent reordering problem include TCP-PR [SBoh04] and [MZha04, HHsi02]. TCP-PR does not rely on Dup ACKs to detect a packet loss, but uses timers to keep track of how long ago a packet was transmitted. pTCP uses its striped connection manager (SM) to handle the TCP re-sequencing while mTCP uses its sub-flow control mechanism for TCP re-sequencing. In PSMC we use a double buffer approach to temporarily hold the out-of-sequence packets, then deliver the in-sequence packets to the TCP handler.

Related works for TCP congestion control in a lossy environment include TCP Westwood [CCas02], which uses the better measured “residual bandwidth” to set TCP congestion window size upon fast retransmit. In PSMC, we use an approximation of the residual bandwidth, not by actually measuring the “residual bandwidth”.

Network Protocols

Figure 2.5 illustrates some commonly-used protocols on OSI seven-layer model.

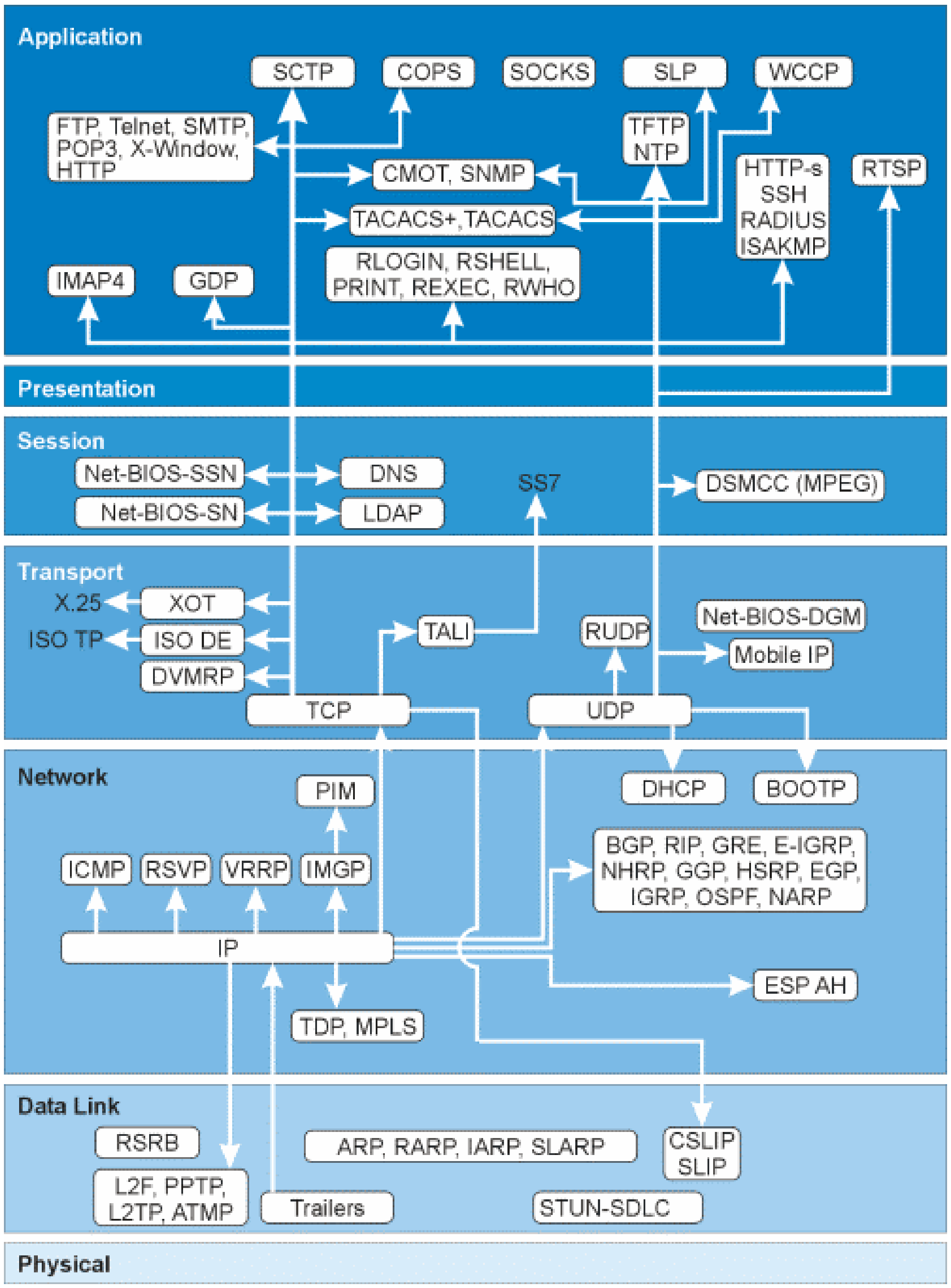


Figure 2.5: Protocols on OSI seven layer [JAna]

IP tunnel

IP is the primary layer-three protocol in the Internet suite. In addition to internet routing, IP provides error reporting and fragmentation / reassembly of datagrams.

We have investigated various approaches to implement indirect routing in PSMC, i.e. SOCKS [SOCK], Zebedee [Zebe], IP Tunnel [IPIP] and IPSec [IPSe].

SOCKS proxy is like an old switch board and can cross wires the connection through the system to another outside connection. SOCKS has several drawbacks. First, it didn’t support UDP, only TCP. Second, it didn’t support certain applications, like FTP. Third, it runs slow.

Zebedee is a simple program to establish an encrypted, compressed “tunnel” for TCP/IP or UDP data transfer between two systems.

IP tunnel (also called IP encapsulation or IP over IP) is a technique to encapsulate IP datagram within IP datagrams (Figure 2.6). This allows datagrams destined for one IP address to be wrapped and redirected to another IP address. The IP tunnel can be set up from Linux to Linux, windows to windows, or between Linux and windows (windows must be Windows 2000 server and above).

The advantages of using IP tunnel are as follows. IP tunnel is a layer three protocol. All the upper layer protocols and applications can utilize it. Second, IP tunnel is a widely used protocol and supported by most modern operating systems. Last but not the least, IP Tunnel itself consumes limited system resources since it is a device descriptor.

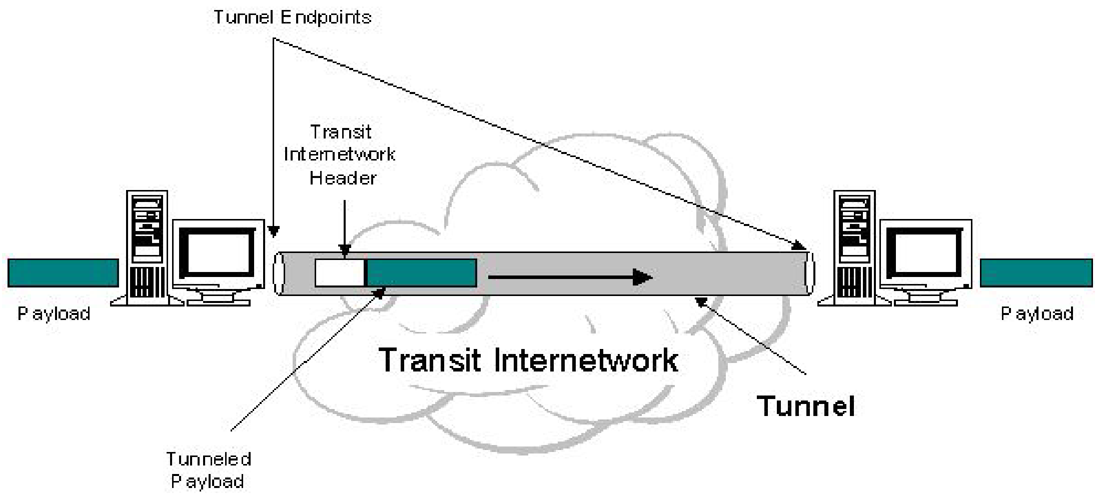


Figure 2.6: IP over IP tunneling [IPIP]

IP Tunnel brings overhead by an extra set of IP headers. Typically it is 20 bytes per packet. So if the normal packet size (MTU) on a network is 1500 bytes, a packet that is sent through a tunnel can only be 1480 bytes big, therefore the payload size is reduced. This also causes fragmentation and reassembly overhead. But these overheads can be reduced or avoided by setting smaller MTU at the client side.

IPSec is an extension to the IP protocol which provides security to the IP and the upper-layer protocols. The IPsec architecture is described in the RFC2401. IPsec uses two different protocols – Authentication Header (AH) and Encapsulating Security Payload (ESP) - to ensure the authentication, integrity and confidentiality of the communication. It can protect either the entire IP datagram or only the upper-layer protocols. The appropriate modes are called tunnel mode and transport mode. In tunnel mode the IP datagram is fully encapsulated by a new IP datagram using the IPsec protocol. In transport mode only the payload of the IP datagram is handled by the IPsec protocol inserting the IPsec header between the IP header and the upper-layer protocol header.

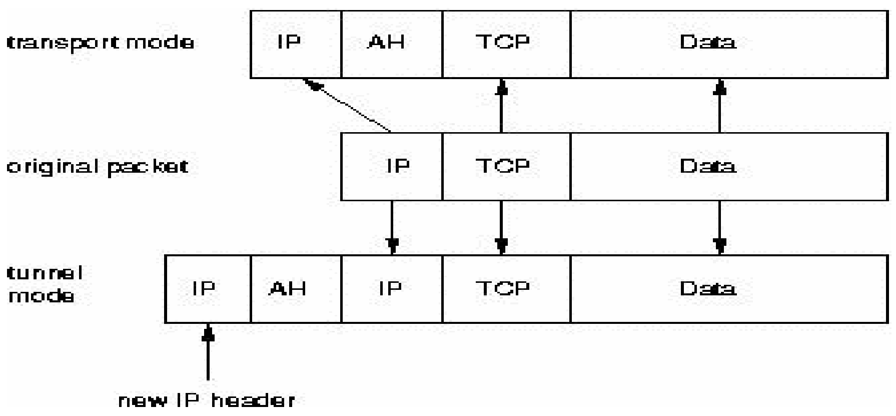


Figure 2.7: IPsec tunnel and transport mode [IPSe]

IPSec and IP tunnel has been used widely in Virtual Private Network (VPN) [VPN]. A VPN is a private network that uses the Internet to securely connect remote sites or users together. Instead of using a dedicated, real-world connection such as a leased line, a VPN uses a "virtual" connection routed through the Internet. From the user's perspective, a VPN operates transparently. The tunneling handshake and packets transmission mechanism in VPN is a good reference for PSMC packets transmission.

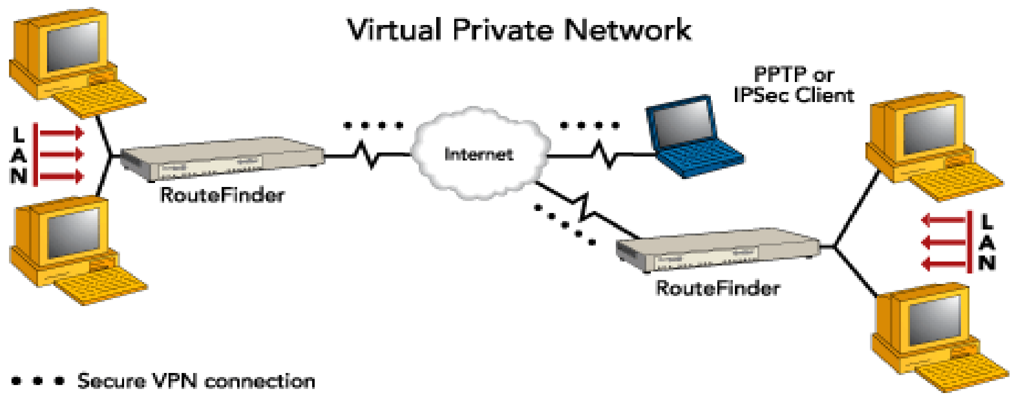


Figure 2.8: VPN [VPN]

TCP

TCP is an end to end protocol which operates over the heterogeneous Internet. TCP has no advance knowledge of the network characteristics, thus it has to adjust its behavior according to the current state of the network. TCP has built in support for congestion control. Congestion control ensures that TCP does not pump data at a rate higher than what the network can handle. For more information on congestion control, please refer to the appendix.

TCP flow control is based on the premise that out-of-order packet is an indication of packet loss, which is not true in multipath environment. Packet loss is detected by Retransmission Time-Out (RTO timer) or Duplicate ACKs (usually three). When Time-out occurs, TCP enters slow start. When dup ACKs occurs, TCP enters fast retransmit and fast recovery.

TCP has four defined congestion control mechanisms to ensure the most efficient use of bandwidth, and quick error and congestion recovery. TCP supports windowing—the process of sending numerous data packets in sequence without waiting for an intervening acknowledgement.

The four mechanisms, defined in detail in RFC 2581, are:

– Slow Start	– Congestion Avoidance
– Fast Retransmit	– Fast Recovery

TCP throughput formula

A simple form is as below:

$$\text{TCP}_{\text{throughput}} = \frac{1.22 * MSS}{RTT\sqrt{p}} \tag{2.1}$$

A more complicated form is as below [JPVF98]:

$$\text{TCP}_{\text{throughput}} = \frac{MSS}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min(1, 3\sqrt{\frac{3bp}{8}})\, p(1+32p^2)} \tag{2.2}$$

Here RTT is Round Trip Time. p is packet lost rate. b is the number of packets that are acknowledged by a received ACK. Many TCP implementations send one cumulative ACK for two consecutive packets received, so b is typically 2. T_0 is the TCP sender times-out.

TCP Implementation in Linux Kernel

In Linux kernel, packets are stored in skbuffs that are sized according to network interface MTU. Kernel-side correspondent for TCP socket is struct sock. struct sock holds state data for the socket (such as the TCP variables regarding congestion window, etc.). There are several queue pointers: outgoing packets not yet acknowledged, incoming packets not yet delivered to application. Queues hold chains of skbuffs. skbuff usually corresponds to one packet sent / received to network. For more information, please refer to the appendix.

UDP

User Datagram Protocol (UDP) is a connectionless protocol that provides the simplest kind of transport services. In keeping with its simple capabilities, the UDP header is short and simple, consisting primarily of a protocol identifier (17) in the IP header, an optional checksum value, an UDP length, and source and destination port addresses.

Appropriate (and historical) uses for UDP concentrate on application layer services that manage their own reliability and connections, such as NFS, and on chatty protocols and services, such as DHCP, SNMP, or RIP that rely on simple controls and fail-safes, and broadcast or periodic transmissions to handle potential reliability, deliverability, or reachability problems. Many multimedia applications and protocols are built on UDP. UDP runs up to 40% faster than TCP under some conditions because of its simplicity.

DDoS, DNS and Overlay

DDoS attacks and DDoS defense mechanisms

The operations of computers and networks rely on the availability of various resources such as network bandwidth, data structures, disk space, and power supply. A consumption DoS attack may be executed against any resource. For example, a TCP half-open (SYN) attack consumes the kernel data structures involved in establishing a TCP network connection. Distributed Denial of Service (DDoS) attacks are any DoS attacks where tools are employed to rapidly “recruit” and coordinate attacks using a mass number of conspirators from widely diverse systems around the globe. Figure 2.9 is a diagram illustrated a typical DDoS attacks.

In general, DDoS defense research can be roughly categorized into three areas: intrusion prevention, intrusion detection, and intrusion response. Intrusion prevention focuses on stopping attacks before attack packets reach the target victim. Intrusion detection explores the various techniques used to detect attack incidents as they occur. Intrusion response research investigates various techniques to handle an attack once the attack is discovered. In addition to these three research areas, intrusion tolerance, once a sub-field of intrusion response, is emerging as a critical research domain.

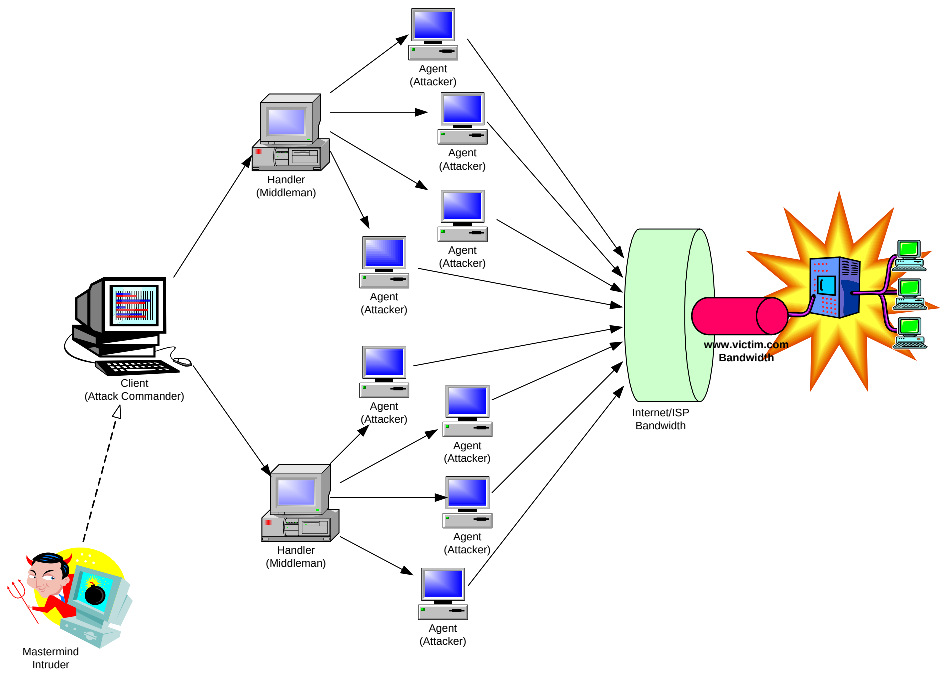


Figure 2.9: A typical DDoS [Chow03]

J. Mirkovic, et al. from UCLA presented taxonomy of DDoS attacks and DDoS Defense Mechanisms [JMir03]. The SCOLD falls into the category of intrusion tolerance and reconfiguration mechanism. Related works in reconfiguration mechanism include reconfigurable overlay networks ([RON01], [DYNA]), resource replication services [JY00] and attack isolation strategies ([BBN]).

The XenoService [JY00] is a distributed network of web hosts that respond to an attack on any one web site by replicating it rapidly and widely. In this way, a mom-andpop antiquarian bookstore that comes under a DDoS attack can within a few seconds acquire more network connectivity than Microsoft, so that it can absorb a packet flood and continue trading.

In [CCac02], Christian Cachin, et al. from IBM presents an intrusion tolerance system named Secure INtrusion-Tolerant Replication Architecture1 (SINTRA). SINTRA supplies a number of group communication primitives, such as binary and multi-valued Byzantine agreement, reliable and consistent broadcast, and an atomic broadcast channel. Atomic broadcast immediately provides secure state-machine replication.

DNS enhancement

DNSSEC [DSEC] (DNS Security Extensions) is one of the major efforts to improve the DNS security. DNSSEC was designed to provide end-to-end authenticity and integrity in DNS. All zone data in DNSSEC is digitally signed with public-key cryptography. By checking the signature, a resolver can verify the validity of a DNS response.

Another major DNS enhancement is dynamic DNS update protocol [DDU], which allows an entity to update a DNS record “on the fly”. Dynamic DNS update can create caching issues and additional problems. Dynamic DNS update was extended to secure DNS update by using a set of keys to authenticate an update [SDU, DSEC]. Digital signatures are stored in the DNS as SIG resource records and are used to encrypt and decrypt update messages for a zone.

DNS has also been extended for purposes other than name-to-address mapping and name resolution. Web server load balancing using DNS, storing IPSec key in DNS, and attribute-base naming system are some of the many examples.

DNS for loading balancing and traffic distribution among a cluster of web servers has been studied in [VCar99, EDDI]. The web servers are known by a single domain name, and DNS dynamically map the domain name to a real web server IP address based on loading balancing algorithm. Therefore, the clients' traffic will be routed to different real server.

In [MRic03], the author proposed a method for storing IPSec keying material in DNS. The IPSECKEY resource record is used to publish a public key that is to be associated with a domain name. It can be the public key of a host, network, or application.

Intentional Naming System [WA99] is a resource discovery and service location system by mapping service name-attributes to name records using an intentional name language.

Overlay network

Overlay network is an area gaining much interest in recent years. The Internet itself is developed as an overlay on the traditional telephone network.

The RON [RON01] is an application layer overlay network that allows distributed Internet applications to detect and recover from path outages and periods of degraded performance within several seconds. It uses UDP encapsulation to send packets along RON nodes. The RON nodes monitor the functioning and quality of the Internet paths among themselves, and use this information to decide whether to route packets directly over the Internet or by way of other RON nodes. RON suffers from scalability problem with more than 50 nodes.

The Detour [SSav99] is an in-kernel packet encapsulation and routing architecture designed to support alternate-hop routing, with an emphasis on high performance packet classification and routing. It uses IP-in-IP encapsulation to send packets along alternate paths. The authors proposed to use intelligent routers spread at key access and interchange points to "tunnel" traffic through the Internet. These intelligent tunnels can improve performance and availability by aggregating traffic information, shaping bursty traffic flows, and using more efficient routes.

Compared with RON and Detour, SCOLD is not only a general purpose overlay network, but also can be used for defending DDoS attacks and improving DNS robustness.

Other overlay networks include the MBone [MBON] for IP multicast, the 6-Bone [IPV6] for IPv6 connectivity and the X-Bone [XBON] for IP-based overlay. X-Bone does not yet support fault-tolerant operation or application-controlled path selection.

Akamai [AKA] is a distributed content delivery system which significantly alleviates service bottlenecks and shutdowns by delivering content from the Internet’s edge. Akamai redirects client requests to the nearest available server likely to have the requested content. The similar between SCOLD and Akamai is that both redirect client traffic. Even though they are used for different purposes, they could benefit from each other by sharing the service servers.

Algorithms for Proxy Server Selection

Cache server selection

Proxy server selection and placement is a critical decision in PSMC. Similar problems, like mirror server and cache server placement and selection problems, are topics gaining interests recent years [EYYM, LQVP01, SJCJ00, PKDR00, BLMG99]. Both mirror server and cache server are used to replicate web content to improve the user-perceived performance and reduce the over-all network traffic.

According to paper [EYYM], there are basically two types of approaches for server selection problem.

Formal approach

It abstracts the network topology to a formal graphic model, and use graphic theory to study the problem. The algorithms are usually based on the following common assumptions:

a) The network topology is pre-known and static.

b) The cost associated with each path is pre-known and static.

c) The network connection between two end nodes is static single path connection.

These assumptions are reasonable for simplifying the network topology, but they are only approximation to the real Internet environment. Vern Paxson has studied extensively the end-to-end Internet dynamics [VPax].

K-center problem is one of the well known optimal server placement problems. For k replicas, we want to find a set of nodes K of size k that allows us to minimize the maximum distance between a node and its closet replica. K-center problem is NP-complete [LQVP01].

The existing formal algorithms include the followings.

a) **Random algorithm**: randomly selecting servers, without consideration of other constrains [LQVP01].

b) **Greedy algorithm**: selecting servers in a greedy fashion and local optimal way [LQVP01].

c) **Tree-based algorithm**: some authors propose solutions by further simplifying the network model from a mesh model to a tree-based model [BLMG99]. However, studies [LQVP01] show that this simplification does not always yield the optimal solution.

d) **K-min algorithm**: by loosing the condition to tolerate the maximum distance between a node and its closest center up to twice the distance of the maximum node-closest center distance, it can be solved in O (N|E|) time [LQVP01, SJCJ00].

e) **Hot Spot algorithm**: place replicas near the clients generating the greatest load [LQVP01].

Practical approach

In real world situation, the network topology and connection costs information might not be pre-known or difficult to obtain. Therefore, the formal approach might not be feasible. There are several practical server selection approaches for real work situation without assumption of pre-known network information. It includes IDMap [SJCJ00] and Client clustering [BKJW00]

IDMap is an architecture designed for global Internet host distance estimation service. It provides a map with Internet distance instead of geographic distance. IDMap utilize a set of Tracers to measure the distance between themselves and Address Prefixes regions of the Internet. Client of IDMap can collect the advertised traces and use them to create distance map.

Client clustering is the approach to cluster the clients and place the web replicas close to the largest concentration of the clients.

Sever selection problem is an extremely difficult problem, and no prevailing approach proposed by far.

Disjoint path selection

The problem of finding disjoint paths in a network has been given much attention in the literature. Various methods have been devised to find a pair of shortest link-disjoint paths with minimal total length [JSRT84, RBha94, JSuu74, RONS89, DSRN91]. In [JSu74], Suurballe proposes an algorithm to find K node-disjoint paths with minimal total length using the path augmentation method. The path augmentation method is originally used to find a maximum flow in a network [CPKS82]. In [JSRT84], the authors improved Suurballe's algorithm such that pairs of link-disjoint paths from one source node to n destination nodes could be efficiently obtained in a single Dijkstra-like computation. In general, this type of problems can be solved in polynomial time [RBha94].

However, similar problems with additional multiple constrains become NP-Complete [GYFK03, ZWJC96, CLSM90]. For example, if requiring the maximal length of the two disjoint paths to be minimized, then the problem becomes NP-Complete [CLSM90]. Heuristic algorithms based on matrix calculation like [EONY95] have been proposed.

An optimal algorithm for finding K-best paths between a pair of nodes is given by Lee and Wu in [SLCW99], where they transfer the K-best paths problem into a maximum network flow and minimum cost network flow algorithm via some modifications to the original graph. Distributed algorithms for the link/node-disjoint paths algorithms can be found in [RONS89].

Complexity

The time complexity of a problem is the number of steps that it takes to solve an instance of the problem, as a function of the size of the input. We generally use Big O notation for complexity to generalize away from the details of a particular computer or implementation. The Big O notation is a mathematical notation used to describe the asymptotic behavior of functions. More exactly, it is used to describe an asymptotic upper bound for the magnitude of a function in terms of another, usually simpler, function.

The complexity class P is the set of decision problems that can be solved by a deterministic machine in polynomial time.

The complexity class NP is the set of decision problems that can be solved by a non-deterministic machine in polynomial time. This class contains many problems that people would like to be able to solve effectively, including the Boolean satisfiability problem, the Hamiltonian path problem and the Vertex cover problem. All the problems in this class have the property that their solutions can be checked effectively.

Differentiated Services

Differentiated Services

The differentiated QoS provisioning problem was first formulated by the Internet Engineering Task Force in the network core. Differentiated Services (DiffServ) [SBDB98] is a major architecture, where the network traffic is divided into a number of classes. It aims to define configurable types of packet forwarding in network core routers, which can provide per-hop differentiated services to per-class aggregates of network traffic.

The proportional differentiation model [CDDS99] states that certain class QoS metrics should be proportional to their pre-specified differentiation weights, independent of the class loads. Due to its inherent differentiation predictability and proportionality fairness, the model has been accepted as an important DiffServ model and been applied in the proportional queueing-delay differentiation (PDD) in packet scheduling [CDDS99, CDDS02, MLJL01, BYPM02, JWCX04] and proportional loss differentiation in packet dropping [YHRG04].

There are recent efforts on differentiation provisioning on end servers [TAKS02, JAMD98, SCCE00, XCPM02, HZHT01]. On the server side, response time is a fundamental performance metric. Existing response time differentiation strategies are mostly based on priority scheduling in combination with admission control and content adaptation [TAKS02, JAMD98, SCCE00].

The work in [XCPM02] adopted priority scheduling strategies, strict or adaptive, to achieve response time differentiation on Internet servers. The results showed that the differentiation can be achieved with requests of higher priority classes receiving lower response time than those of lower priority classes.

However, this kind of strategies cannot quantitatively control quality spacings, say proportionally, among the classes. Time-dependent priority scheduling algorithms developed for PDD provisioning in packet networks can be tailored for PDD provisioning on Web servers [SLJL04]. However, they are not applicable for proportional response time differentiation because the response time is not only dependent on a job's queueing delay but also on its service time, which varies significantly depending on the requested services. Providing proportional response time differentiation on Web servers is not only important, but also challenging.

There are efforts on the design of new resource management mechanisms at kernel level to support Diff-Serv provisioning efficiently, as exemplified by resource containers [GBPD99], and its extension cluster reserves [MAPD00].

Resource container is a new operating system abstraction. It separates the notion of a protection domain from that of a resource principal. A resource container encompasses all system resources that the server uses to perform an independent activity, such as processing a client HTTP request. All user and kernel level processing for an activity is charged to the appropriate resource container and scheduled at the priority of the container. Resource containers allow accurate accounting and scheduling of resources consumed on behalf of a single client request or a class of client requests.

Thus, this new mechanism can help provide fine-grained resource management for DiffServ provisioning when combined with an appropriate resource scheduler. However, while kernel-level mechanisms can provide efficient control over resource management, their weaknesses lie on the portability and deployment issues.

Proportional differentiation

The proportional differentiation model was proposed in the network core [CDDS99]. It was first applied for DiffServ provisioning in packet scheduling and packet dropping, in which packet queueing delay and loss rate are key QoS factors, respectively. Many algorithms have been designed to achieve proportional delay differentiation (PDD) in the network routers.

They can be classified into three categories: rate-based; see BPR [CDDS99] for example, time-dependent priority based; see WTP [CDDS02] and adaptive WTP [MLJL01] for examples, and Little's Law-based; see PAD [CDDS02] and LAD [JWCX04] for examples. The work in [CLJL04] demonstrated that some of the algorithms can be tailored for request scheduling for PDD provisioning on the server side. However, the algorithms are not applicable to proportional response time differentiation because response time is not only dependent on a job's queueing delay but also on its service time, which varies significantly depending on the requested services.

In [XCPM02, MTMS04], the authors addressed priority-based request scheduling strategies for response time differentiation on Web servers. Incoming requests were categorized into the appropriate queues with different priority levels for the corresponding services. Requests were then executed according to their strict priority levels [XCPM02] or adaptive priority levels [MTMS04]. The results showed that response time differentiation can be achieved in the sense that higher classes receive less response time than lower classes. However, the quality spacings among different classes cannot be guaranteed by the priority scheduling strategies. Therefore, this kind of priority-based scheduling strategies cannot achieve proportional response time differentiation on Web servers.

Our integrated approach improves over the previous efforts in the sense that it can quantitatively control quality spacings between different classes and provide robust proportionality of response time differentiation.

In [XZJW04], the authors proposed a processing rate allocation strategy for server-side DiffServ provisioning in terms of slowdown in E-Commerce applications. They left a challenging implementation issue; that is, how to practically achieve the processing rate for various traffic classes on servers.

In [HZHT01], the authors adopted an M/M/1 queueing model to guide node-based resource allocation for stretch factor (a variant of slowdown) DiffServ provisioning in a server cluster. However, to achieve the processing rates for different classes, the node partitioning strategy still needs the support of resource allocation on individual servers.

In this thesis, we design and implement a practical application-level process allocation approach on an Apache Web server to achieve differentiated processing rates.

In [TAKS02], the authors utilized feedback control approaches to achieve overload protection and performance guarantees on Web servers. The strategy was based on real-time scheduling theory which states that response time can be guaranteed if server utilization is maintained below a pre-computed bound. Thus, control-theoretical approaches, in combination with content adaptation strategies, were formulated to keep server utilization at or below the bound.

In this thesis, we design and integrate a PID feedback controller with the queueing-theoretical rate allocation. Our approach is complementary to the previous work in the sense that our approach integrates the queueing theory and control theory for proportional response time differentiation.

CHAPTER III

PROXY SERVER BASED OVERLAY NETWORK

In this chapter, we present the design and implementation of a proxy server based overlay network called the Secure Collective Defense (SCOLD) system. SCOLD is a general purpose application layer overlay network. It can be used to defend against DDoS attacks and to provide alternate or backup routes.

Introduction

DDoS attacks exploit a number of compromised machines and launch large coordinated packet floods towards a target, thereby causing denial of service for legitimate users. DDoS attacks have been an immense threat to the Internet for years. One of the most prominent attacks recently is on Akamai [AKA] in June 2004 that creates major Akamai and Internet DNS Problems.

The increasing frequency and severity of network attacks reveal some fundamental security problems of today's Internet. The Internet was designed to provide fast, simple and reliable communication mechanisms, and its tremendous success is a credit to the original design. However, many network services like DNS and protocols like TCP/IP were not designed with security as one of the basic considerations. Also, the highly distributed and interdependent nature of Internet provides opportunities and resources for the coordinated and simultaneous attacks by malicious participants. Due to the same nature of Internet, it is difficult to enforce common security policies, measurements and coordination among the participants of Internet. Therefore, the existing Internet architecture needs to be strengthened and services / protocols need to be enhanced or re-designed with security in focus.

In this chapter, we present a novel DDoS defense system called Secure COLlective Defense (SCOLD) system. The key idea of SCOLD is to follow intrusion tolerance paradigm by providing clients with alternate routes via a set of proxy servers and alternate gateways when the normal route is unavailable or unstable due to DDoS attacks, network failure or congestion. The main techniques utilized in SCOLD are the enhanced Secure DNS Update and Indirect Route [Chow04, DWil04]. SCOLD can also be used as a general purpose application layer overlay network.

In SCOLD, the enhanced DNS system is utilized to store and convey the indirect routing information, including the set of proxy server IP addresses. There are two steps to enable the indirect routing in SCOLD. First, the client DNS server needs to get the indirect routing information from the target DNS server. This is accomplished by the enhanced secure DNS update. Second, after clients get the indirect routing information from the client DNS server, clients can set up indirect route to the target server. Thus the communication channels between clients and the target are kept open by using indirect routes during DDoS attacks.

System Overview

Motivation

Most organizations today deploy multiple gateways or multi-homing scheme [AAJP04] as a backup measure in case of network congestion or failure. Recently overlay network [SSav99, RON01] has been developed for the same purpose. When the main gateway is congested or unavailable due to DDoS attacks, the legitimate traffic should be redirected through the alternate gateways. However, the alternate gateways are exposed to public. They are subjected to DDoS attacks too. Therefore, simply adding more alternate gateways may not be sufficient to defend DDoS attacks.

Most existing DDoS defense mechanisms presume the scenario where packets are transmitted along a normal Internet route and via the main gateway. Under very large-scale DDoS attack, the huge volume of attack traffic at the main gateway will consume most of the available network resources. Techniques like rate-limiting [TGMP01] and filtering [MAZU] which are performed behind the main gateway will become less effective. Other technique such as traceback [DSAP01, SS00] may require support from upstream routers and still being developed as protocol standards.

The SCOLD system defends against DDoS attacks by setting up indirect routes between clients and target server. The traffic between clients and target server is transported over Internet through the indirect routes. In SCOLD, the three main problems that need to be solved are as follows.

a) How to redirect the heterogeneous clients' traffic through indirect route?

b) How to utilize alternate gateways while hiding their IP addresses from public domain?

c) How to prevent the attack traffic from using indirect route?

We solve the first problem by setting up indirect route via a collection of geographically separated proxy servers and alternate gateways. We solve the second and third problem by using proxy servers that are equipped with IDS, firewall and rate-limiting mechanism, and only expose the IP addresses of the proxy servers to the public clients.

System architecture

Figures 3.1-3.3 illustrates how the SCOLD system works. Figure 3.1 shows a target site under DDoS attacks where R is the main gateway, and R1-R3 are the alternate gateways. In the figure the majority of the traffic from net-a.com is malicious, that of net-b.com is legitimate, and that of net-c.com is mixed.

Figure 3.2 shows the control flow of the SCOLD system. When the target site is under DDoS attacks, its Intrusion Detection System (IDS) raises an intrusion alert and notifies the SCOLD coordinator, who sits in the same or trusted domain of the target server. The coordinator selects a set of proxy servers between the clients and the target server, and notifies the selected proxy servers, proxies 2 and 3 here, to set up indirect routes. The proxy servers notify the DNS servers of the client networks to perform a secure DNS update. The clients from net-b.com and net-c.com are notified with indirect route, but net-a.com is not notified due to its malicious traffic pattern which is detected by the IDS on the target network.

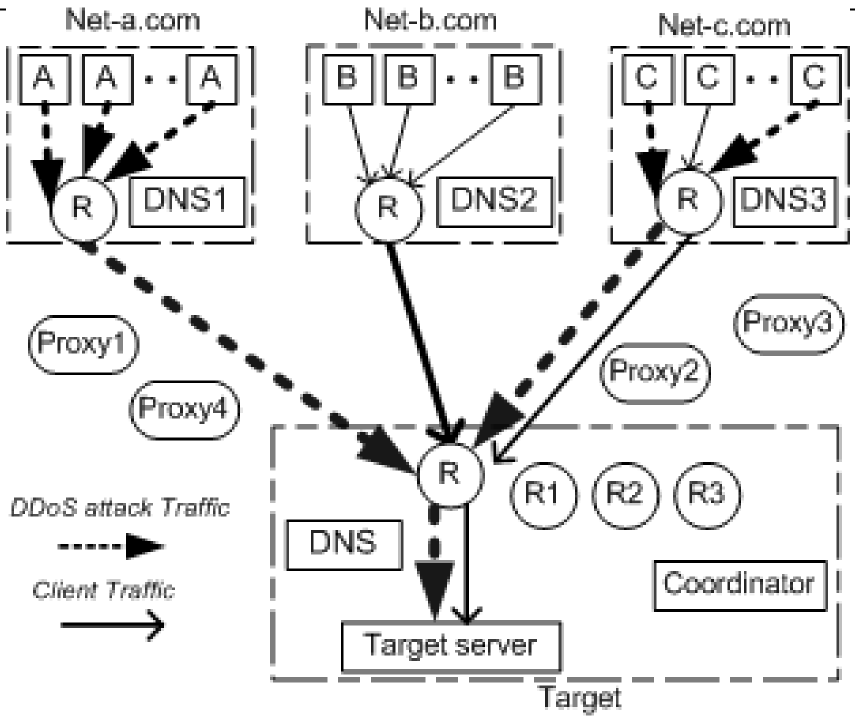


Figure 3.1: Target site under DDoS attack

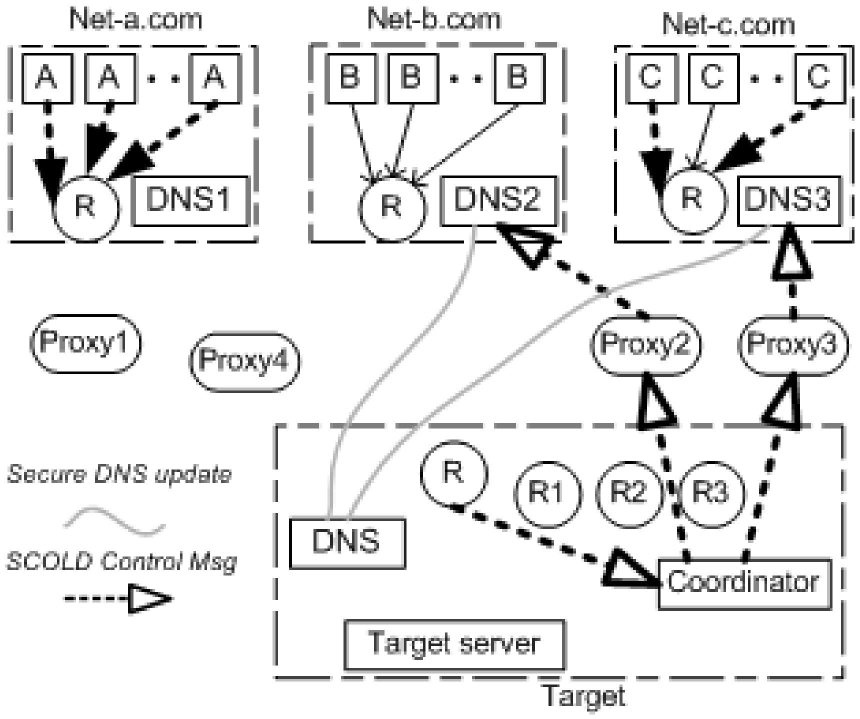


Figure 3.2: The control flow in SCOLD

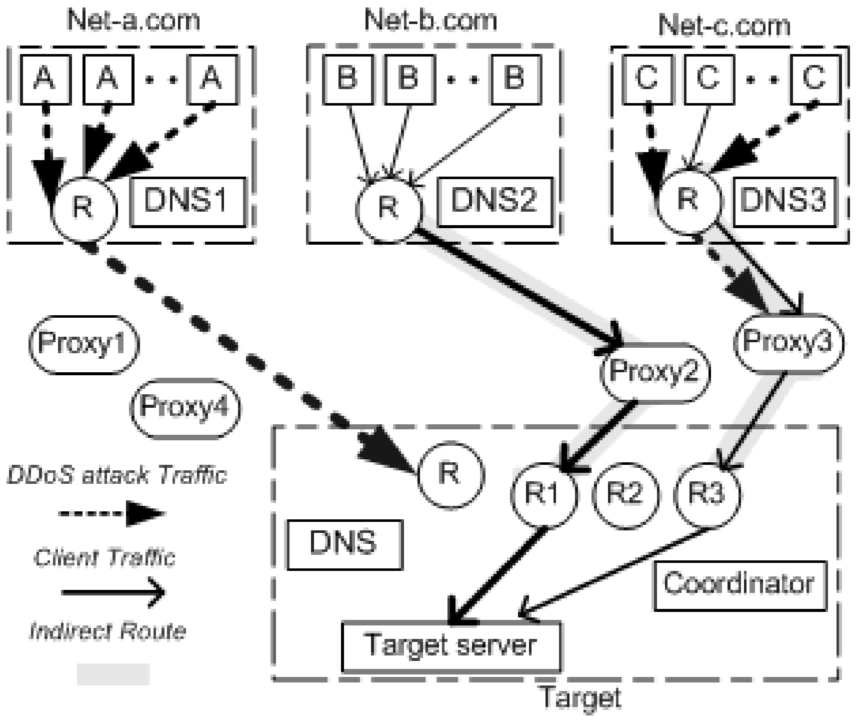


Figure 3.3: Indirect route in SCOLD

Figure 3.3 shows how an indirect route is setup in the SCOLD system. After a secure DNS update, the client side DNS server gets the new DNS entry containing the designated proxy server IP addresses. The clients query their DNS server, get the set of proxy server IP addresses, and set up indirect routes to the target server via the selected proxy servers. The proxy servers examine the incoming traffic and relay it to the designated alternate gateway on the target site.

On the client side, the name resolve library needs to be enhanced to support the indirect routing. In enterprise environment, the internal clients go outside through an enterprise gateway (or an enterprise proxy server). Instead of modifying the client resolver, the enterprise gateway (or the enterprise proxy server) needs to be enhanced to support the indirect route.

In SCOLD, the IP addresses of the alternate gateways and the SCOLD coordinator(s) are revealed only to the trustworthy proxy servers to protect them from being attacked by malicious clients. The clients in public domain can connect to the target side through the designed proxy servers. To avoid traffic analysis at the proxy servers by intruders, multiple proxy servers can be deployed in a chain on an indirect route.

The proxy servers in SCOLD are enhanced with IDS and firewall filters to block malicious traffic that may try to come in through the indirect route. The detection of intrusion on the proxy servers can provide additional information for identifying and isolating the spoofed attack sources. In Figure 3.3, by combing the distributed intrusion detection results from the main gateway R and the proxy server 3, the attack source from net-c.com could be more accurately identified.

A proxy server itself may suffer from DDoS attacks or get congested when large volume of traffic comes through it. Assuming a large collection of proxy servers available, the impact of heavy traffic can be alleviated by spreading traffic over multiple proxy servers.

The procedure for resuming normal route is similar to setting up indirect route. The proxy servers need to notify the client DNS servers with another secure DNS update to restore the normal DNS records. The clients query the DNS server and start to resume the normal direct route. We can also set an “expiration time” for indirect route so that SCOLD can automatically revoke obsolete indirect routes.

All the control messages communicated in SCOLD system are encrypted using Secure Sockets Layer (SSL) and all nodes involved must be mutually authenticated. Experiments show that this is one of the major causes of overhead in SCOLD system.

Proxy servers can be provided by the participating organizations of SCOLD, or fee-based service providers, like Akamai [AKA].

Note that different proxy server selection may result in different system performance; and multiple proxy servers can be selected to enable parallel transmission or multi-path connection. We study these problems in [YCai05].

More SCOLD applications

Enhanced SCOLD proxy servers with bandwidth throttling can be used to defend large-scale DDoS attacks. The SCOLD coordinator collects and analyzes the target server system load, available network bandwidth and the statistics of the client traffic. Based on the information, the coordinator can decide what the allowed maximum bandwidth is for each proxy server connecting to the target server. The proxy servers equipped with admission control and rate-limiting mechanism can enforce such bandwidth throttling. In Figure 3.3, the coordinator may assign different allowed maximum bandwidth to proxy 2 and 3, depending on the sever load and client behavior. This integrated IDS can help to control aggressive or malicious clients and reserve resources for normal operation.

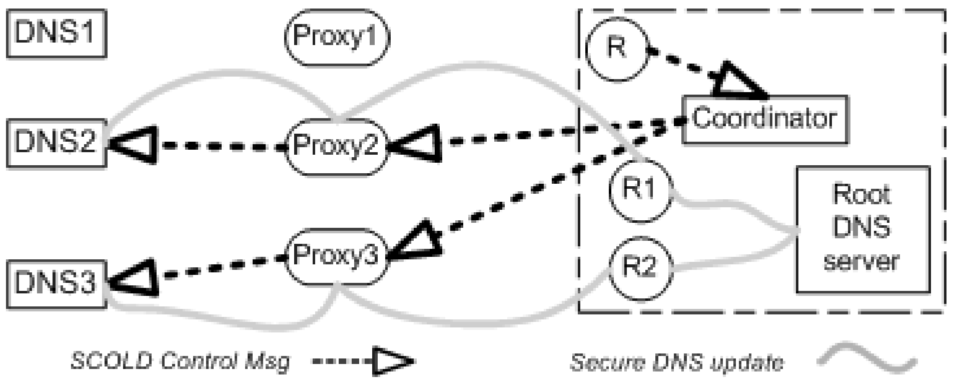


Figure 3.4: Protect the root DNS server

A slightly revised version of SCOLD can be used to protect the Root DNS servers from DDoS attacks, like the one caused a brief service disruption on the nine of the thirteen DNS root servers in 2002 [NEWS-1]. In Figure 3.4, DNS 1-3 are the client side DNS servers, and the main gateway R of the root DNS server is under sever DDoS attacks. DNS 1-3 may experience significant delay or even failure when querying the root DNS server. Due to the current DNS querying model, the end users will perceive a poor Internet performance with unbearable delay.

By utilizing the SCOLD technique, we can set up indirect routes between client DNS and root DNS to ensure the normal operation of root DNS server. The IDS on the root DNS server raises alert and notifies the coordinator; the coordinator notifies the selected proxy servers (proxy 2, 3 here); the proxy servers notify the legitimate client DNS servers with their IP addresses; those DNS servers then set up indirect routes to the root DNS via the proxy servers and the alternate gateways; then the client DNS servers can query the root DNS server via indirect route.

In SCOLD architecture, the proxy servers become the “frontline” fighting against the DDoS attacks. It brings several benefits. First, with large number of proxy servers available, the target server gain more resources to defend DDoS attacks. Second, if a proxy server fails, we can quickly recruit other proxy servers without significant lost. Third, proxy servers with integrated IDS can provide powerful functionalities to detect and defect attacks.

In SCOLD, there are three defense lines against DDoS attacks. First, based on the preliminary intrusion detection result from the main gateway, some malicious clients will not be notified with indirect route. Second, the proxy servers are equipped with IDS and firewall filters to further block malicious traffic. Third, the proxy servers are equipped with admission control and rate-limiting mechanism to enforce bandwidth throttling and control the aggressive clients.

Enhanced Secure DNS Update

In SCOLD, the DNS is utilized to store and convey the indirect routing information, which are the proxy server IP addresses. This requires several modifications and enhancements on current DNS.

First, we need to redefine the DNS record format for storing the additional information. A sample of the new DNS record in the DNS zone file looks like the following.

```
target.targetnet.com.  10  IN  A    133.41.96.71
target.targetnet.com.  10  IN  ALT  203.55.57.102
                       10  IN  ALT  203.55.57.103
                       10  IN  ALT  185.11.16.49
```

The first line is a normal DNS entry, containing host name and its IP address. The next 3 lines contain the IP addresses of proxy servers, as the newly defined “ALT” type (type 99).

The DNS zone data needs be securely updated from the target side DNS server to the client side DNS server upon request. However, in the scenario of DDoS attack, the main gateway of the target server domain may become unavailable or unstable. Therefore, the DNS update might experience significant delay or even failure. By setting up indirect route and perform the DNS update via the indirect route, we can overcome the problem.

Figure 3.5 illustrates how the enhanced DNS update works. Step 1, the target side IDS raises intrusion alert, and notifies the coordinator. Step 2, the coordinator notifies the selected proxy server(s). Step 3, the proxy server notifies the client DNS server for a secure DNS update. Step 4, if the client DNS server decide to make a DNS update, it sends a request back to the proxy server for setting up indirect route; if the proxy server grants the permission, it notifies a selected alternate gateway and the target server for setting up indirect route; then an indirect route from the target DNS server to the client DNS server via the proxy server and the alternate gateway is set up. Step 5, the client DNS server performs the secure DNS update and gets DNS zone records from the target DNS server.

In the enhanced DNS update, we can not only update DNS zone file through indirect route, but also perform DNS query through indirect route. In Figure 3.5, after the indirect route is set up, the client DNS server can query the target DNS server through the indirect route, without being affected by the DDoS attacks.

Indirect Route

We investigate several alternatives for implementing indirect route, including SOCKS proxy [SOCKS], Zebedee [ZEBE], IPSec [IPSE] and IP tunnel [IPIP]. SOCKS proxy server is like an old switchboard and can cross wire between connections. The main drawbacks of SOCKS are that it doesn’t support UDP and FTP. Zebedee is an application to establish an encrypted and compressed tunnel between two systems. But it requires specific configuration per network application. IP tunnel is a technique to encapsulate IP datagram within IP datagram. This allows datagram destined for one IP address to be wrapped and redirected to another IP address. IP tunnel provides what we want for indirect route. IPSec is an extension to the IP protocol which provides security to the IP and the upper-layer protocols. We believe whether client traffic needs to be encrypted is a client decision. Therefore, we choose IP tunnel to support basic indirect routing. However, the implementation using IP tunnel can be migrated to using IPSec easily. IP tunnel and IPSec have been used widely in Virtual Private Network (VPN) [VPN] to set up “tunnel” between network nodes and redirect traffic.

The advantages of using IP tunnel are as follows. IP tunnel is a layer three protocol. All the upper layer protocols and applications can utilize it. Second, IP tunnel is a widely used protocol and supported by most modern operating systems. Last but not the least, IP Tunnel itself consumes limited system resources since it is a device descriptor.

There is overhead associated with IP Tunnel due to the extra set of IP header and the reduced payload size. This can also cause fragmentation and reassembly overhead. In our experiments, the overhead in term of response time varies between 30% and 200%. But compared with the impact of DDoS attack, which may cause unbearable delay, the overhead of IP tunnel is still in an acceptable range. Fragmentation overhead can be avoided if we restrict the message transfer size at the sender.

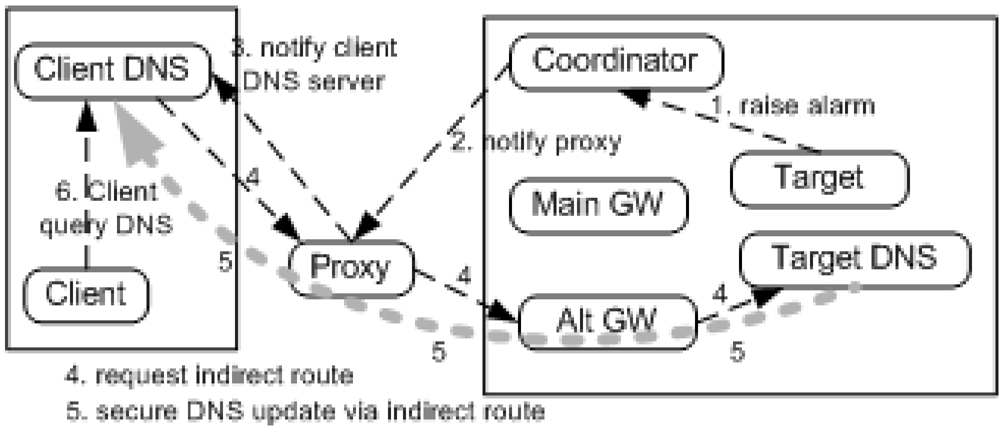


Figure 3.5: Secure DNS update via indirect route

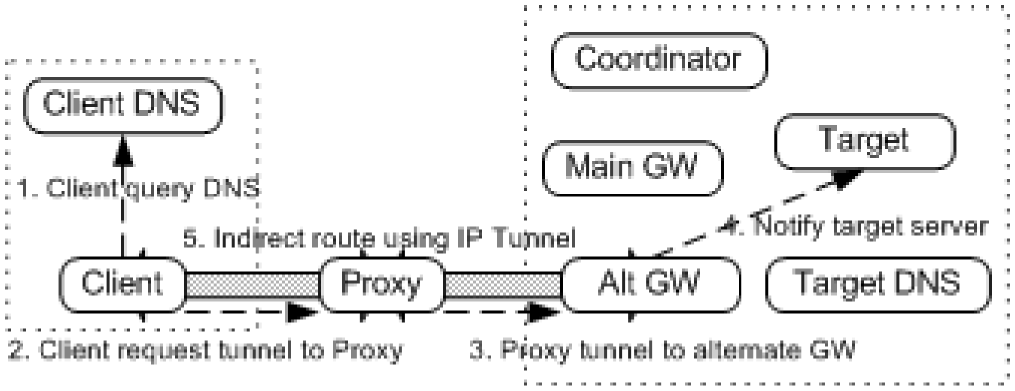


Figure 3.6: Indirect route by using IP tunnel

Figure 3.6 illustrates how the indirect route set up by using IP tunnel. The client queries its DNS and get the IP addresses of proxy servers; the client sends a request to a proxy server for indirect route; if the proxy server grants permission, it notifies the designated alternate gateway; the alternate gateway notifies the target server, then an indirect route can be set up between the client and the target server via the proxy server and the alternate gateway. We set a timeout value at client side in case the communication is lost or the indirect route is broken.

Implementation

Implementation summary

Our implementation on BIND 9 and Redhat Linux 8 / 9 is summarized as follows.

1) The BIND9 (v.9.2.2) DNS server [BIND9] was modified to support the newly defined ALT type 99 data and to enable the automated secure DNS update.

2) The DNS dynamic update utility (nsupdate [NSUP]) was enhanced to support indirect routing and the new data type. The enhanced DNS update utility is named nsreroute.

3) On client side, the domain name resolve library (v.2.3.2) was enhanced to support the new data type and enable the automated set up of indirect route. In Redhat Linux, the resolve library is usually located in /usr/lib or /lib directory, and named as libresolv-nnn.so (nnn is the version). The routing table on the client node needs to be modified at run time.

4) An agent program runs on the participating nodes (client DNS server, target DNS server, proxy server, alternate gateway and target server) listening for the control message. The routing table on the participating node needs to be modified at run time.

5) The indirect route is implemented by using IP Tunnel [IPIP]. By modifying the routing table at run time, we can utilize IP tunnel just like normal Ethernet devices, like eth0. We also tested indirect route on Windows 2000 server using IP tunnel

6) All the control messages are encrypted using Secure Sockets Layer (SSL) and all participating nodes must be mutually authenticated. The implementation of authentication and encryption/decryption mechanism is a difficult decision, especially in large-scale distributed system. However, this is not the key focus of the chapter. We utilize the most commonly-used public key cryptography and digital certificate in OpenSSL (v.0.9.6) [OSSL].

Enhanced Resolve Library

In Redhat Linux, the resolve library is usually located in /usr/lib or /lib directory, and named as libresolv-nnn.so (nnn is the version). The source code of resolve library can be obtained from glibc package. We modify the res_query.c file under glibc/resolv directory (version 2.3.2). The source code is listed briefly below.

```
int  __libc_res_nquery (){
      static int scold_count = 0;
      // scold_count is used to prevent multiple callings of SCOLD in one session.
      …..
      ….
      if(scold_count ++  < 1){  //not to run multiple times in one session
            check_result = check_target_status (target_server_name);
            /* target_server_name is the name of the target server,
            * check_result return value: if 0 means keep current settings;
            * -1 means to clear all tunnels and n means setup n tunnels.*/

            if(check_result != 0 && check_result != -1){ //set up tunnel now
              setup_IPTunnel(target_server_name);
            }
            else if (check_result == -1){ //clear existing tunnels
              clean_IPTunnel(target_server_name);
            }
            else if (check_result == 0){//keep current settings
            }
          }
        return (n);
}
/*user defined functions*/
int check_target_status (char *server_name) {
        /*scold daemon on client update scold_status file upon indirect routing requests*/
        status = read_scold_status_file (server_name) ; //read in the scold_status file
        if (scold_timeout(server_name))
                status = -1; //timeout occurs, overwrite status to clean all tunnels
        return status;
}
void setup_IPTunnel(char *server_name) {
        /*scold daemon on client update scold_proxy file with proxy server IPs*/
        proxy_server_list = read_scold_proxy_file (server_name); //read in proxy IP addresses
        _setup_IPTunnel (proxy_server_list); //set up IP tunnel via the proxy servers
}
void clean_IPTunnel(char *server_name) {
        proxy_server_list = read_scold_proxy_file (server_name); //read in proxy IP addresses
        _clean_IPTunnel (proxy_server_list); //clean IP tunnel via the proxy servers
}
```

Experimental and Simulation Results

In this section, we present some experimental and simulation results on SCOLD.

Experimental setup

We set up a test bed consists of more than 20 nodes with various machine settings. The test bed includes HP Vectra machines (PIII 500MHz, 256MB RAM, 100Mb Ethernet connection), HP Kayak machines (PII 233MHz, 96MB RAM, 10/100 Mb Ethernet connection), Dell machines (PIII 1GHz, 528MB RAM, 100 Ethernet connection) and virtual machines (96MB RAM, 100 Mb virtual Ethernet connection, running on a Dell machine with dual PIII 1.2GHz and 4G RAM). The operating systems are Linux Redhat 8, 9 and Windows 2000 server. StacheldrahtV4 [STA4] is used as the DDoS attack tool. Figure 3.7 is one of the test beds which we used in the experiments.

Analysis of the experimental results

a) SCOLD initial setup overhead.

We first evaluate the time taken to initially set up an indirect route in SCOLD, which is the SCOLD initial setup overhead. As discussed previously, there are three steps involved. Step 1, "IDS -> coordinator -> proxy". The overhead comes from the secure communication among nodes. Step 2, "Proxy -> client DNS -> perform secure DNS update". The overhead comes from the secure communication and the secure DNS update. Step 3, "client -> client DNS -> set up indirect route". The overhead comes from the secure communication, the client side resolve library processing overhead and the time to set up indirect route.

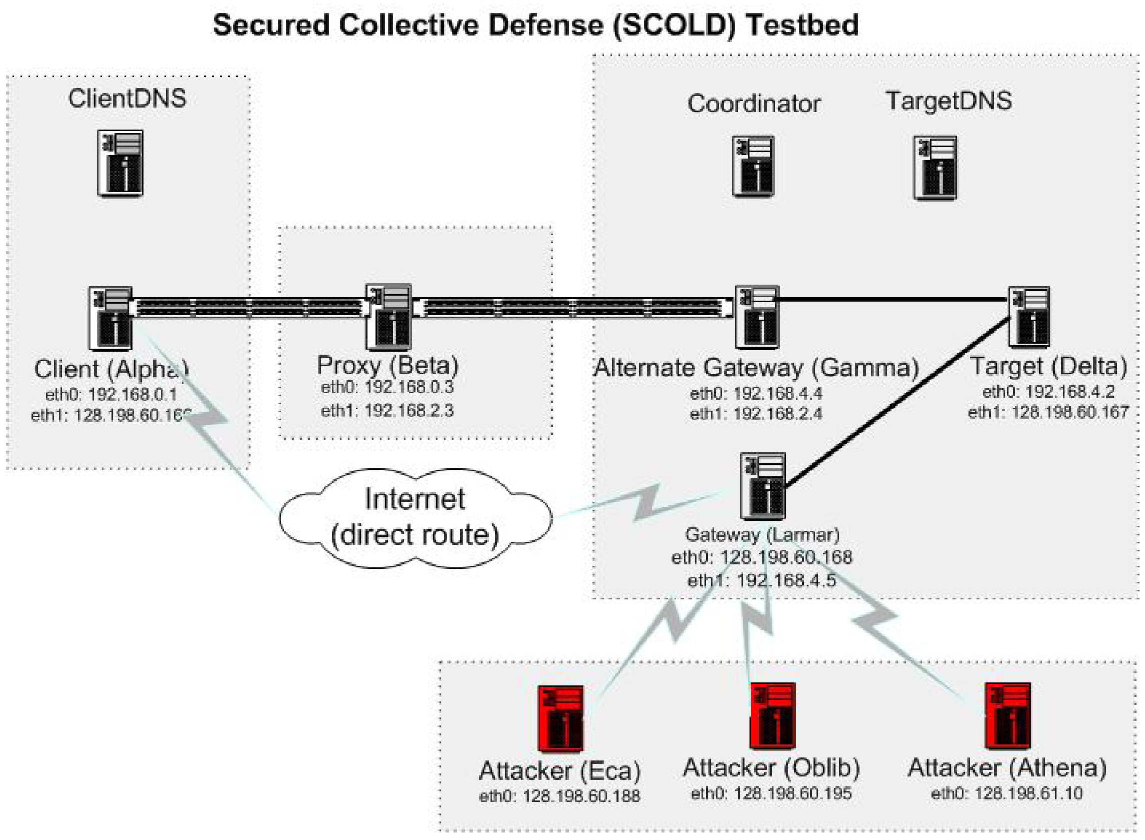


Figure 3.7: SCOLD testbed

Table 3.1 shows the initial setup time in SCOLD. It is observed that the overhead comes primarily from the secure DNS update and the secure communication among nodes. Table 3.2 further shows that the secure DNS update time increases dramatically when the number of client DNS servers increase. This suggests that there is a limit on how many client DNS servers a proxy server can handle concurrently.

Table 3.1: SCOLD initial setup time (second)

Step 1	Step 2	Step 3	Total
2.1	4.7	2.7	9.5

Table 3.2: Secure DNS update time (second)

1 DNS	10 DNS	25 DNS	50 DNS
4.7	25	96	240

b) SCOLD performance

Next we evaluate the SCOLD performance. Table 3.3 shows the processing overhead of using indirect route vs. the possible delay of direct route under DDoS attacks. The SCOLD processing overhead comes from the IP tunneling overhead and more Internet hops involved in indirect route. We can observe that the overhead of indirect route in term of response time is about 70%. Further experiments shows the overhead varies from 30%–200%. However, under DDoS attack, the response time of using direct route increases dramatically (15 times to infinity), while the response time of using indirect route keep the same (No DDoS attacks against proxy servers directly in the tests, same below). Table 3.3 also shows that the SCOLD performance is relatively independent of the application type (Ping, HTTP, FTP).

Table 3.3: Indirect Route processing overhead vs. Direct Route delay under DDoS attack

Test	No attack		Under DDoS attack			
	Direct Route (a)	Indirect Route (b)	Direct Route (c)	Indirect Route (d)	Direct Route Delay (c) / (a)	Indirect Route Overhead (b - a) / (a)
Ping	49 ms	87 ms	1048 ms	87 ms	21 times	77%
HTTP(100k)	6.1s	11s	109s	11s	18 times	80%
HTTP(500k)	41s	71s	658s	71s	16 times	73%
HTTP(1M)	92 s	158s	timeout	158s	infinity	71%
FTP(100k)	4.2 s	7.5s	67s	7.5s	16 times	78%
FTP(500k)	23 s	39s	345s	39s	15 times	69%
FTP(1M)	52 s	88s	871s	88s	17 times	69%

We also evaluate the performance the enhanced secure DNS update. Table 3.4 shows performance comparison between an enhanced DNS update with indirect route (using nsreroute) vs. normal secure DNS update with direct route (using nsupdate). It shows that the nsreroute with indirect route is usually slower than the nsupdate with direct route by 30 - 70%. The overhead is mainly caused by the time to set up indirect route and transport DNS data via indirect route. However, when the main gateway of the target site is under DDoS attack, the nsupdate with direct route is impacted seriously, and the nsreroute with indirect route is almost not affected.

Table 3.4: Performance of nsreroute vs. nsupdate,

with and without DDoS attack

	No attack		Under DDoS attack			
	nsupdate (a)	nsreroute (b)	nsupdate (c)	nsreroute (d)	nsupdate Delay (c) / (a)	nsreroute Overhead (b-a) / (a)
1 DNS	4.2 s	7.1s	50 s	7.1 s	12 times	70%
10 DNS	21.1 s	27.4 s	timeout	27.4 s	infinity	30%

c) More overhead analysis

Table 3.5 shows the overhead of enhanced resolver. We measure the response time to resolve a domain name by using enhanced resolver and the original resolver. It is observed that the enhanced resolve library only imposes very limited overhead compared to original resolve library.

We evaluate the overhead of the enhanced BIND DNS server. Table 3.6 shows the response time to answer a domain name query by using the enhanced DNS server and the original DNS server. The result shows that the overhead of the enhanced DNS server is also very limited.

Table 3.7 shows the overhead of IP tunnels itself. It is observed that the number of IP tunnels on network nodes doesn't affect the performance, because IP tunnel itself consumes very limited system resources.

Table 3.5: Performance of enhanced resolver vs. original resolver

Test	Enhanced resolver	Original Resolver
Ping	0.7 ms	0.6 ms
HTTP	0.7 ms	0.7 ms
FTP	0.7 ms	0.7 ms

Table 3.6: Performance of enhanced DNS vs. original DNS

Test	Enhanced DNS	Original DNS
Ping	1.2 ms	1.1 ms
HTTP	1.2 ms	1.1 ms
FTP	1.2 ms	1.1 ms

Table 3.7: The influence of how many tunnels exist

Test	1 tunnel	10 tunnels	50 tunnels	100 tunnels
Ping	87 ms	87 ms	87 ms	87 ms
HTTP(100k)	11s	11s	11s	11s

The alternate route via the proxy server may be under DDoS attacks too. Therefore we may want to give up the current alternate route and recruit a new proxy server to set up a new alternate route in some cases (assuming route initialization and DNS update already finish). Table 3.8 shows how long it takes to remove the current route and recruit a new route dynamically. In the test, we start a long web downloading task via a selected proxy server. Then we launch a DDoS attack against the selected proxy server. The sender notices a significant delay on the current alternate route. After a timeout period (set to 90 second in the test), the sender decides to give up the current route and recruits a new route. It takes about 2.8 seconds to finish a route deletion or route addition.

Table 3.8: Path detection, deletion and addition

Action	Time to finish (second)
Route Delete	2.8
Route Add	2.8

Simulation results

To further analyze the overhead in SCOLD, the ns2 simulator [NS2] was used to perform the simulation study for large-scale network. The topologies used in simulation are generated using GT-ITM [GITM]. We create transit-stub graphs with 100-500 nodes. We pick nodes in the same stub for target server, target DNS server, coordinator, main gateway and 3 alternate gateways. We randomly pick 10% nodes as proxy servers, 5% nodes as DDoS attackers, 20% nodes as clients and 4% nodes as client DNS servers.

For simplicity, we set the overhead of IP tunneling and the overhead of secure communication to be a fixed percentage with a small random variance. We randomly generate background traffic whose average is 60% of the total network bandwidth. We generate DDoS attack traffic which can completely shutdown the victim. We keep proxy servers away from being attacked directly.

Figure 3.8 shows that the average initial setup time of indirect route increases slowly when

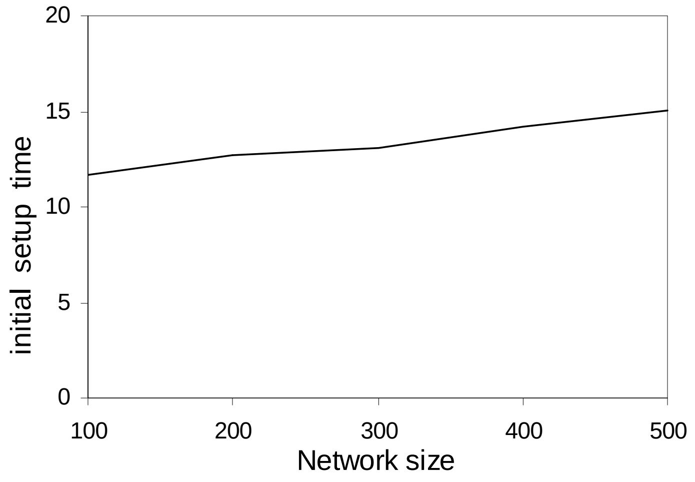


Figure 3.8: average initial setup time vs. network size

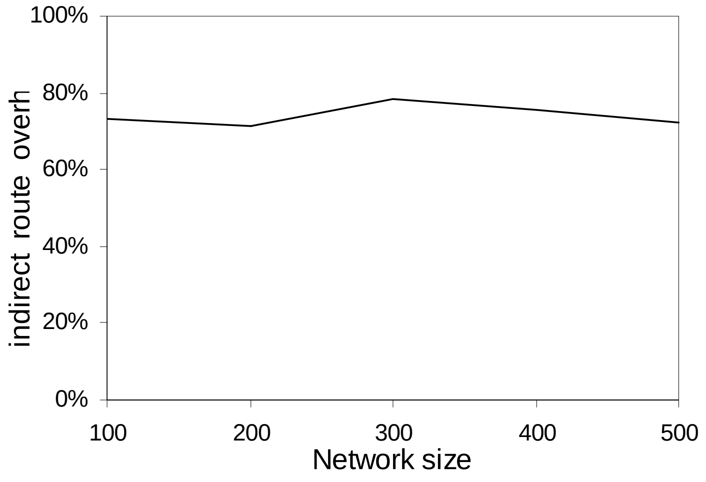


Figure 3.9: indirect route processing overhead vs. network size

the network size increases. Figure 3.9 shows that the indirect route processing overhead keeps nearly constant when the network size increases. In both figures, SCOLD demonstrates good scalability with respect to the initial setup overhead and the processing overhead.

Conclusion

SCOLD redirects the traffic between clients and servers through indirect routes via proxy servers and alternate gateways. BIND9 DNS package and its secure DNS update utility were enhanced to support indirect route. IP tunnel was utilized to implement indirect routing. The results show that SCOLD can improve the network security, availability, and performance.

SCOLD raises several issues. First, how should the Internet community form trust relationships and coordinate with each other. Second, how to detect and deal with the compromised proxy server nodes. These are important research issues and go beyond the scope of this chapter.

It is our hope that the research results of SCOLD can produce valuable secure software packages, and provide insights for network security and Internet cooperation.

CHAPTER IV

PROXY SERVER BASED MULTIPATH CONNECTION

In this chapter, we present the design and implementation of a multiple path approach named Proxy Server based Multipath Connection (PSMC), which can utilize multiple network paths in parallel and aggregate the available bandwidth of these paths. The TCP/IP protocol is enhanced to support multipath connection.

Introduction

One key challenge in today's Internet is to improve network performance, security and reliability for various network users. The current network connection is mostly over a single path connection, which may under-utilize network resources and suffer from performance problems. It is also vulnerable to potential attacks, link breakage or even traffic congestion.

Multipath connection provides potential multiple paths between network hosts. The traffic from a source is spread over multiple paths and transmitted in parallel through the network. Multipath connection not only can improve the network performance, but also cope well with network congestion, link breakage and potential attacks.

In this chapter, we present a new multiple path connection approach named Proxy Server based Multipath Connection (PSMC). We address a number of challenging key issues in developing a multipath system as discussed below.

a) How to set up multiple paths between two end hosts. In PSMC, we set up multiple paths via a set of intermediate connection relay proxy servers by using IP tunneling. The mechanism is based on SCOLD system [Chow04, DWil04].

b) How to stripe packets across multiple paths. In PSMC, the packet striping is done in the IP layer in a weighted round robin manner. Both TCP and UDP can benefit from PSMC.

c) TCP persistent reordering problem [SBoh04]. TCP packets over multiple paths are likely to reach the destination out of sequence order. Our experimental results show that it can seriously degrade the overall system performance. In PSMC, we use a double buffer at the TCP layer on the receiver side to solve the problem.

d) TCP high loss rate problem. The loss rate of a multipath connection is usually higher than that of a single path connection. Traditional TCP blindly cuts the congestion control window size in half upon fast retransmit, which may slow down the TCP performance in multipath scenario. In PSMC, we set the congestion window size to a more appropriate value upon fast retransmit.

e) "Bad" path detection. Experimental results show that a failed path, a "bad" path, or paths with "uneven bandwidth distribution" can seriously affect the system performance. In PSMC, by passively monitoring on end hosts and periodically exchanging network information through the communication channel, we can quickly detect the unwanted paths.

f) Path management and Failure recovery. To achieve maximum aggregate bandwidth, algorithms need to be designed to select the best paths. At the same time, path addition and path deletion need to be done dynamically with low cost in a timely manner. The multipath system should also recover quickly from a path failure.

g) In PSMC, we propose a **communication channel** between sender and receiver for exchanging network traffic information.

We implement PSMC on a network of Linux systems. The experimental results show that our approach can significantly improve the network aggregate bandwidth, network security and reliability.

Background

Multipath connection is a topic gaining interest. Early works include Maxemchuk [NMax75] in 1975. In the network layer, multipath connection has been studied extensively under the name of multipath routing [SLee00, SVut01, JChe98]. These routing schemes require changes on intermediate routers, which may limit the usage.

PSMC sets up multiple paths based on the SCOLD system [Chow04, DWil04]. The SCOLD is essentially an overlay network. Previous works on overlay-based techniques include Detour [Deto] and RON [RON01]. Overlay network is a feasible solution for multipath connection by utilizing the existing Internet infrastructure. For example, mTCP [MZha04] is built on RON network. However, RON suffers from a scalability problem with more than 50 nodes [RON01]. SCOLD has better scalability and flexibility [Chow04].

The closest multipath schemes to our work are mTCP [MZha04] and pTCP [HHsi02]. Both are limited to TCP only, while PSMC supports TCP as well as UDP. Another major difference is that PSMC can be installed on one end-host (one-way multipath) or on two end-hosts (two-way multipath). In the first case, only the data packets from sender are spread out over multiple paths, the return ACK packets from receiver still go through the main direct path. In the second case, both the forwarding packets and the return packets are sent through multiple paths. pTCP is designed to support only one-way multipath.

Packet striping can occur on a different layer. The application layer [THac02, HSiv00] and data link layer [HAdi96, I802] implementations suffer from the inability to accurately profile the available bandwidth on individual paths. The TCP layer implementations like mTCP and pTCP use a different striping scheme by monitoring and keeping track of the outstanding packets on each path, which may impose operational overhead and a complicated mechanism.

Previous works for TCP persistent reordering problem include TCP-PR [SBoh04] and [MZha04, HHsi02]. TCP-PR does not rely on Dup ACKs to detect a packet loss, but uses timers to keep track of how long ago a packet was transmitted. pTCP uses its striped connection manager (SM) to handle the TCP re-sequencing while mTCP uses its sub-flow control mechanism for TCP re-sequencing. In PSMC we use a double buffer approach to temporarily hold the out-of-sequence packets, then deliver the in-sequence packets to the TCP handler.

Related works for TCP congestion control in a lossy environment include TCP Westwood [CCas02], which uses the better measured “residual bandwidth” to set TCP congestion window size upon fast retransmit. In PSMC, we use an approximation of the residual bandwidth, not by actually measuring the “residual bandwidth”.

PSMC Design

System Overview

Figure 4.1 is a diagram that illustrates the overview of a PSMC system. The multipath sender module is responsible for packet distribution among the selected multiple paths. Some packets will go through the normal “direct route”, others might go through the alternate “indirect routes” depending on the packet distribution. The intermediate connection relay proxy servers examine the incoming packets and forward to the destinations through the selected path. The multipath receiver module collects the packets arrived from multiple paths, reassembles them in order and delivers to the upper layer.

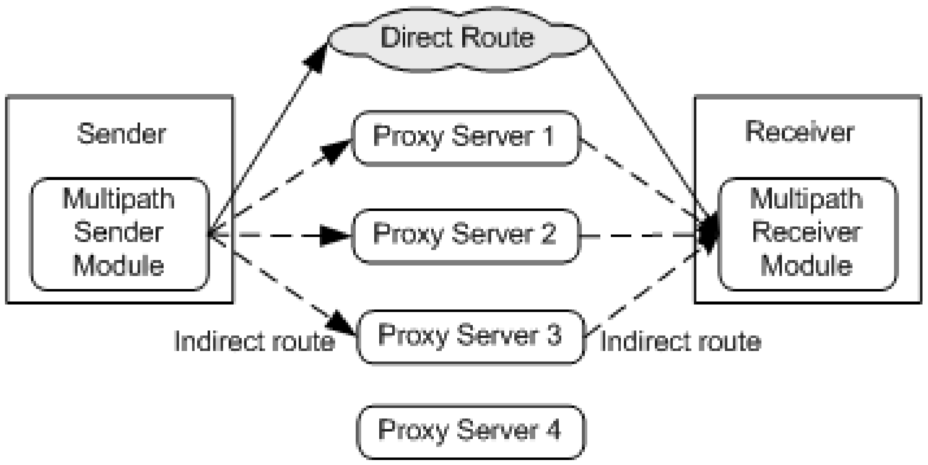


Figure 4.1: Proxy server based multipath connection (PSMC)

In [SSav99], the authors find out that the default Internet direct route is usually not the best. There exist many alternate routes which are much better. These findings justify the usage of indirect routes.

For a network connection, sometimes the bottleneck lies at edge of the network, say the telephone line for a dial-up modem user; or the bottleneck is the host processing power, say the web server capacity. This may limit the benefits of using multipath connection. However the situation is under significant improvement with the fast development of technology. In this chapter we assume that the network bottlenecks are not in the above two scenarios.

PSMC can improve network reliability and availability with multiple redundant paths available. In [TNgu03, ABan96], the author proposed to use multiple paths to send redundant error correction information to improve the transmission reliability. PSMC can also be used to improve network security. For example, IPSec [IPSE] in real-time multimedia service becomes feasible with adequate bandwidth available. PSMC makes certain attacks like DDoS attacks, traffic analysis and traffic hijack harder to succeed, since the traffic is spread over multiple paths.

The establishment of multiple paths in PSMC is based on the Secure Collective Defense (SCOLD) system [Chow04, DWil04]. SCOLD originally is designed to defend against the DDoS attacks, but it can also be used to provide multiple alternate paths between two end hosts via a set of proxy servers. The proxy servers can be provided by the participating organizations of SCOLD, or fee-based service providers. The indirect routes are based on IP tunneling.

Some of the notations in this chapter are summarized as follows. Assuming there are n paths between two end hosts, the available bandwidth, loss rate, Round Trip Time (RTT) and one way delay (OWD) on path i is noted as BWi, pi, RTTi, OWDi respectively, i=1…n.

Packet Striping on the IP layer

We decide to stripe packets on the IP layer because we want both TCP and UDP to benefit from multipath connection. It is no doubt that TCP is used by the majority of current Internet applications. However, most TCP flows on Internet are small [YZha02]. They do not gain as much benefit from multipath connection as large TCP flows do. On the other hand, the ever-increasing demands on real-time multimedia service from Internet gain UDP more and more interest. Many audio/video transmission protocols like [VOIP, RSTP, RSVP] prefer to UDP. And most applications are the long-lived ones with a large amount of packets to transmit. Therefore, we believe supporting UDP is important when designing a multipath mechanism.

In PSMC, we adopt a weighted round robin data striping scheme among multiple paths. To achieve the maximum usage of available bandwidth, the data striping ratio should be the same as the ratio of available bandwidth on each path. For example, there are two paths of 10Mb and 5Mb, then the data striping ratio should be 2:1. With the traffic information from the communication channel, the sender can dynamically and more accurately adjust its data striping ratio. For example, the sender’s data striping ratio is 2:1 while the receiver’s data receiving ratio is 3:2, the sender should adjust its striping ratio to 3:2.

Double buffer for TCP re-sequencing

Multipath connection brings a problem of TCP packet “persistent reordering” problem. Because the individual paths have different latencies, packets over multiple paths are likely to reach destination out-of-order. The TCP receiver observes the received packet sequence numbers, and generates duplicate ACK (Dup ACK) for each out-of-sequence segment. After the sender receives three Dup ACKs, it will enter fast retransmit. TCP fast retransmit is based on the premise that out-of-order packet is an indication of packet loss, which is not true in the multipath environment. Our experimental results show that without considering the packet reordering, the aggregate bandwidth of multipath connection might be even worse than that of a single path connection in extreme cases.

The rational of our double buffer approach is as follows. Most out-of-order packets in multipath connection are not caused by packet loss. Therefore, a better approach is to hold the packet in a temporary buffer (not the TCP receive buffer), and wait for the expected packet to arrive. If the expected packet arrives in time (before buffer is full), then we deliver the in-sequence segment to the TCP handler. If the buffer is full and the expected packet still doesn’t arrive, this indicates a likely real packet loss. We then deliver some out-of-sequence packets to TCP handler and let TCP send a few Dup ACKs to trigger a fast retransmit.

The size of double buffer needs to be carefully considered. If the buffer size is too small, then it can not hold all the waiting packets. If the buffer size is too large, then it takes too long to trigger a fast retransmit for a real packet loss. Obviously the buffer size upper limit is the congestion window size cwnd, otherwise the packet flow may halt.

Now we try to derive the buffer size lower limit. Assume path 1 has the largest one way delay (OWD) among all paths, OWD1 = OWD_{max}. In the initialization stage of the buffer, at time OWD 1, the first packet on path 1 arrive the destination. On path i, there are already $((OWD_{\max} - OWD_i) * BW_i + 1)$ packets arrive the destination (For simplicity, here BWi is in terms of packets/second. We also assume no packet striping delay on sender). All these packets need to be hold in the buffer. So the minimum buffer size is

$$B1 = \sum_{i=1}^{n} ((OWD_{\max} - OWD_i) * BW_i + 1) \tag{4.1}$$

In the steady stage of packet transmission, in a time unit (second), there are BWi packets arrived on path i. Assuming path k has the smallest bandwidth, $BW_k = BW_{min}$. For the time period of $1/BW_{min}$, path k has two consecutive packets arrived. On path i, at most $(BW_i / BW_{\min} + 1)$ packets arrived. So the minimum buffer size in this case is

$$B2 = \sum_{i=1}^{n} (BW_i / BW_{\min} + 1) \tag{4.2}$$

The buffer size lower limit should be the maximum of B1 and B2.

$$B = \max\left\{ \sum_{i=1}^{n} ((OWD_{\max} - OWD_i) * BW_i + 1),\ \sum_{i=1}^{n} (BW_i / BW_{\min} + 1) \right\} \tag{4.3}$$

We observe from above that the buffer size is related to the difference of the paths' latency and bandwidth. A fast link combined with a slow link, or a large capacity link with a small capacity link, may need a bigger buffer size than two moderate links. This may slow down the overall performance. Therefore, we suggest that paths with similar characteristics should be selected. This path selection criterion is made possible with a large number of geographically diverse proxy servers available.

In the double buffer approach, the incoming packet is placed on the buffer in order of the sequence number (earliest sequence number are closest to the head) and packets with duplicate sequence numbers are removed. When the packet with the correct sequence number arrives, the in-sequence packet segment is fed to the TCP processing code. Otherwise, the buffering continues.

If the packet is lost, the expected packet will never arrive. There are several solutions for the missing packet.

1) The double buffer does nothing. The TCP timer will time out and send another acknowledgment requesting the packet. This is similar to RTO and slow stat. It is too conservative and doesn’t perform well.

2) Use a fixed-size double buffer scheme. With a predefined double buffer size, when the double buffer is full, then delivers the first 3 packets in double buffer to TCP handler to trigger a fast retransmit. However, how to set double buffer size correctly and dynamically is a problem. Our experimental results show that a fixed size double buffer is not able to adapt well to the dynamic traffic condition.

3) The third solution is to dynamically adjust the size of the double buffer. We propose an adaptive TCP double buffer algorithm below. The notations in the algorithm are as follows: dbsizeup (double buffer size up limit), dbsizelow (double buffer size

lower limit), dbsize (double buffer size), N (the number of paths), packet_in_sequence (the number of in-sequence packets delivered to TCP handler)

Adaptive TCP double buffer algorithm

```
1) Buffer initialization,
        dbsizelow = max {N, user input value},
        dbsizeup = min{ssthresh, user input value},
        dbsize = a * dbsizelow (a is a user specified parameter, usually a=2)
        packet_in_sequence=0
2) Check the incoming packet and put in double buffer,
3) Check double buffer for in-order sequence segment and deliver this segment to TCP handler if it exists.
        packet_in_sequence += segment size
4) If double buffer is full,
        deliver first 3 packets in double buffer to TCP handler to trigger a fast retransmit,
        dbsize ← dbsize -1 if dbsize > dbsizelow,
        packet_in_sequence=0.
5) If packet_in_sequence > dbsize,
        dbsize ← dbsize +1 if dbsize < dbsizeup,
6) loop back to step 2.
7) For each sampling period, or RTO timeout, dynamically update
        dbsizelow = formula (4.3),
        dbsizeup = cwnd,
```

dbsize = a*dbsizelow

TCP congestion window control

Another TCP related issue in multipath connection is the TCP congestion control window size. In multipath environment, the packet loss rate on the aggregate paths is:

$$(1-\prod_{i=1}^{n}(1-p_i)).$$

In PSMC, all paths share the same TCP congestion window. When three Dup ACKs are detected, instead of blandly reducing the congestion window size in half, we should adjust it to the "Residual Bandwidth", which is equal to the difference between the total bandwidth and the bandwidth of the path causing packet loss.

Assuming on the sender side, the packet striping ratio is s_i, (i=1...n). If path k is the route which causes packet loss, then the "residual bandwidth" is:

$$(\sum_{i=1}^{n} s_i - s_k)/\sum_{i=1}^{n} s_i * cwnd.$$

As discussed in Section 3.4, we usually select paths with similar characteristic. Therefore in practice, we adopt a simpler approach by set the "residual bandwidth" as

((n-1) / n) * cwnd

Our experimental results show that the approximation is acceptable.

The new TCP congestion window control algorithm is as follow:

TCP congestion window control algorithm

When three duplicated ACKs are detected,

1) set ssthresh = ((n-1) / n) * cwnd, (instead of 1/2*cwnd in TCP Reno);

2) if (cwnd > sshthresh) then set cwnd = sshthresh;

3) The rest is the same as TCP Reno or new Reno.

Communication channel and passive monitoring

The network information like bandwidth, loss rate and latency for individual paths is important for decision making in PSMC. Active probing on the network is usually undesirable since it imposes extra load on the network and interferes with normal traffic. Passive monitoring on end host is usually acceptable. However, keeping track of each sub-flow as in mtcp may impose overhead and suffer from scalability problem when the number of paths increases.

In PSMC, we perform passive monitoring on the end hosts to collect the network information on the whole traffic flow. A secure communication channel is set up between the end hosts to periodically exchange the information.

Path management

By using networking measurement tools, we can estimate the available bandwidth on network links and get the network topology. Extensive works have been done in this field [KLai01]. Then the well-known labeling algorithm for maximum network flow problem can be used to find the maximum aggregate bandwidth between two end hosts and select the best proxy servers to set up alternate routes [FGlo92].

PSMC may encounter the problem of path failure or “bad” path. Our experimental results show that “bad” paths (meaning paths with extremely uneven bandwidth distribution) can dramatically degrade the overall system performance. Therefore, bad paths and failed paths need to be removed from the PSMC routing list.

Taking bandwidth as a performance metric (latency and error rate will be similar), if the metric of path i is significantly below the average of other paths, we treat it as a “bad” path. Our experimental results suggest the threshold to be 1:10.

When a path is removed from the routing list, the end hosts will stop using this path. However, the packets which were already sent out through this path are unaffected.

The end hosts can also dynamically add new paths to the routing list. When the sender observes that the current aggregate bandwidth drops significantly, it can start probing other proxy servers and choose some of them to set up new paths.

Note that even with higher packet loss rate and higher path failure probability, PSMC can still improve the network reliability and robustness. Neither packet loss nor sub-path failure will stop the whole traffic flow. The probability of all sub-path failure is much lower than that of single path failure. Our experimental results show that PSMC can recover quickly from a path failure.

PSMC on UDP

PSMC on UDP is much simpler than on TCP. The congestion control, packet persistent reordering and transmission errors are usually handled by the UDP application itself.

However, there are a couple of things that need to be addressed. Multipath connection increases the available bandwidth with a price of higher loss rate, and the loss rate increases when the number of routes increases. This is sometimes unacceptable to multimedia protocols. Redundant or error correction information can be sent over multiple routes for packet loss [TNgu03].

The second issue is that UDP is an "aggressive" protocol without built-in transmission rate control and congestion control mechanism. A congested link that is only running TCP is approximately fair to all users. However, when UDP data is introduced into the link, there is no requirement for the UDP data rates to back off, forcing the remaining TCP connections to back off even further. This is unfair to TCP.

One possible solution is to set a UDP transmission rate upper limit on each path at end hosts to control the aggressiveness of UDP. This UDP upper limit can be negotiated between end hosts through the communication channel according to the application requirements.

Implementation

Implementation summary

The PSMC system is implemented on the Linux kernel 2.4.24, and can be migrated to other kernel versions. We create a kernel patch which provides the necessary kernel environment information and interface to the PSMC modules. The kernel modification is primarily on net/ipv4/ip_output.c, tcp_input.c and tcp.c.

The whole PSMC system is designed into several independent modules based on functionality. The PSMC is a loosely-coupled system. These modules can be loaded, unloaded and maintained dynamically upon operational request. This greatly enhances the system scalability, maintainability and flexibility.

The interface between PSMC modules and end users is through the /proc file system. End users can conveniently input the PSMC parameters (like buffer size, routing information and data striping ratio) through the /proc system. The communication channel is built on secure socket connection with OpenSSL [OSSL].

IP striping

PSMC packet striping is implemented on IP layer. We modify the ip_output.c file under the linux/net/ipv4 directory. To modulate the code, we put the PSMC functional code in a PSMC module, and insert a function pointer in the ip_output.c.

```
//insert a function pointer to scold module in ip_output.c

int (*scold_function) (struct sk_buff *) = 0 ;

int ip_queue_xmit(struct sk_buff *skb) {
…
…
        if (scold_function && (*scold_function)(skb)) {
            // the return value contains the new device information for packet striping
            // linux will automatically do the packet striping with the new device
    }
```

```
        return NF_HOOK(PF_INET, NF_IP_LOCAL_OUT, skb, NULL, rt->u.dst.dev,
                    ip_queue_xmit2);
}
```

We also declare the inserted function pointer in netsyms.c under linux/net/ipv4 directory.

```
    //declare and export the function in netsyms.c
    extern int (*scold_function) (struct sk_buff *);
    EXPORT_SYMBOL_NOVERS (scold_function);
```

In the PSMC module, we implement a weighted round robin packet dispatcher. Other packet scheduling schemes can be easily implemented as well. The code is briefly listed below.

```
// weighted round robin packet dispatcher for psmc
int  psmc_wr_dispatcher(struct sk_buff *skb) {
    /* psmc daemon updates the psmc_wr file with
    * destination, weight, proxy server IPs and tunnel device information.
    * psmc_dest is the destination address */
    read_psmc_wr_file (psmc_dest); //read in psmc_wr file

    //get the current position in the weight round robin dispatching manner
    current_position = weighted_round_robin (psmc_dest);
    //get the current device to dispatch packet
    dev_m = dev_get_by_name (get_device (current_position));

    if (dev_m == NULL){
            return 0; //no such device, use original device
    }
    if (dev_m){ //dev exists
            return psmc_dev[current_position];
            //return value is the tunnel device number to send packet
            //linux will automatically send packet via this new device
    }
  }
  return 0; //use original device
}
```

Double buffering

TCP Persistent reordering problem is a major hit on the network performance. A solution to out of order packets is to queue the arriving packets and wait for the delayed packet. This packet buffering is done using the INET socket. The INET socket is connection based, so there is one INET socket per connection and is a structure containing information about the connection (e.g,. sequence number).

If the expected packet is lost, then it will never arrive. There are several solutions for missing packet.

1) The double buffer does nothing. The TCP timer will time out and send another acknowledgment requesting the packet.

2) Use a fixed-size double buffer.

3) Use an adaptive double buffer algorithm by dynamically adjust the size of the double buffer.

The adaptive algorithm performs better than algorithm 1 and 2. More efficient double buffer algorithms can to be designed and analyzed in future.

Our implementation of adaptive double buffer is listed below. We first redefine the struct sock by adding a scoldLog struct, which is similar to backlog. The scoldLog is the double buffer which temporarily holds the packets between IP and TCP.

```
//modify the sock.h under linux/include/net directory
struct sock {
…..
   struct {
        struct sk_buff *head;
        struct sk_buff *tail;
   } scoldLog;
}
```

Then we modify the tcp_ipv4.c file under linux/net/ipv4 directory to implement the double buffer algorithm.

```
// modify the tcp_ipv4.c file under linux/net/ipv4 directory

int tcp_v4_rcv(struct sk_buff *skb)

{

…

  bh_lock_sock(sk);

  ret = 0;

/*The psmc scold buffer interacts with end user through proc file system.

* End user can turn on/off buffer and set buffer size via proc.

* packets in scold buffer are sorted with sequence number from small to large

* for simplicity, we list the code in high level*/

  if (scold_buffer_on) { // if turn on the buffer

        insert_packet_scold_buffer (); // insert packet into sorted scold buffer

        // if there is a packet segment in order, deliver the in-order segment to tcp handler

        if (segment_in_order_scold_buffer()){

             deliver_segment_scold_buffer ();

            delivered_packet_in_sequence += segment size;

            // if many packet delivered in sequence, increase dbsize

            if (delivered_packet_in_sequence > dbsize && dbsize < dbsizeup)

                  dbsize++;

        }

        //if double buffer is full, deliver first three packets to tcp handler to trigger fast retransmit

        if(scold_buffer_full()){
```

```
        deliever_first_three_packets();
        if (dbsize > dbsizelow)      dbsize --;
        packet_in_sequence=0;
    }
    //periodically update dbsize, dbsizelow and dbsizeup
    update_db_size(sample_period);
}
…
}
```

Experimental Results

Experiment setup

We set up a testbed that consists of more than 20 nodes. Table 4.1 lists the machine setup.

Table 4.1: Machine setup in the testbed

	Hardware	OS
HP Vectra	PIII 500MHz, 256MB RAM, 100Mb/s Ethernet connection	Redhat 9
HP Kayak	PII 233MHz, 96MB RAM, 10/100 Mb/s Ethernet connection	Fedora Core 1
Dell	PIII 1GHz, 512MB RAM, 100Mb/s Ethernet connection	Fedora Core 1
VMWare virtual machine	96MB RAM, 100 Mb/s virtual Ethernet connection, running on a Dell machine with dual PIII 1.2GHz and 4GB RAM	Fedora Core 1

Analysis of the experimental results

a) PSMC performance analysis on throughput

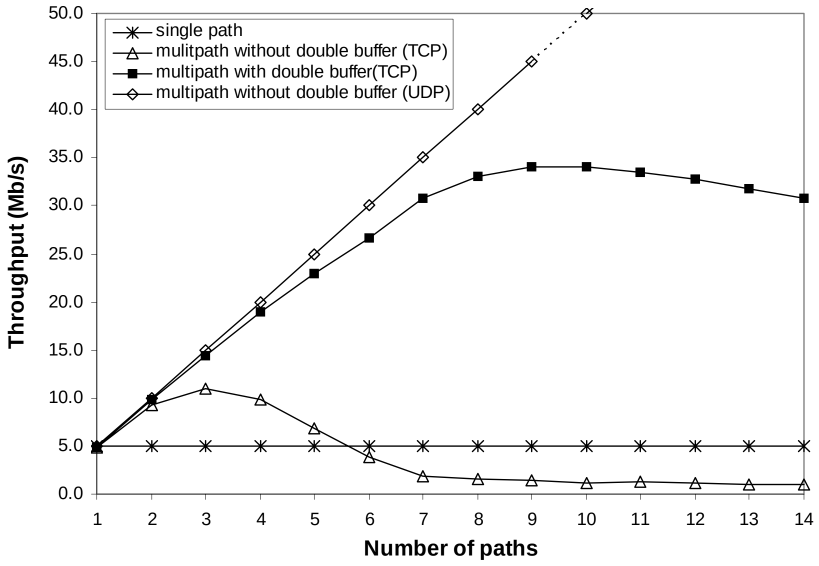


Figure 4.2a: PSMC throughput comparison

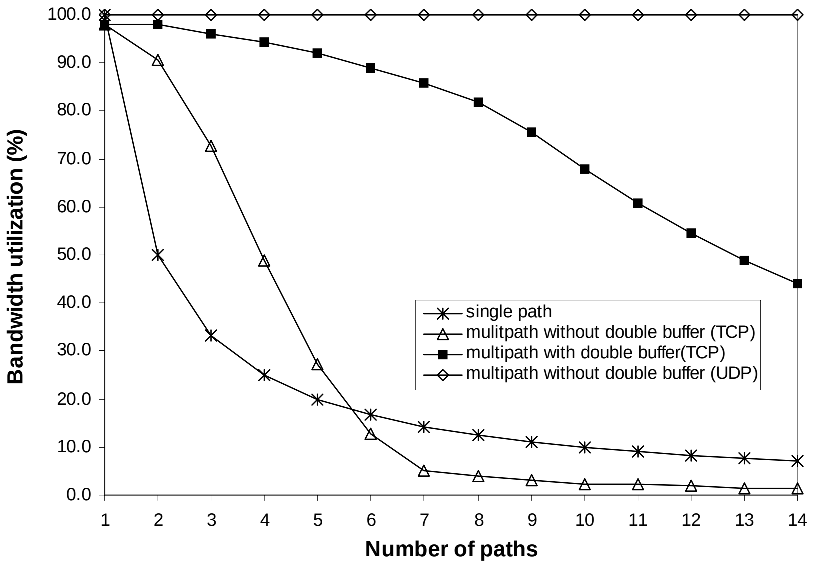


Figure 4.2b: PSMC bandwidth utilization comparison

Figure 4.2a shows the throughput comparison between single path connection, multipath without double buffer (TCP), multipath with double buffer (TCP) and multipath without double buffer (UDP). The x axis is the number of paths in use. The y axis is the measured throughput. The bandwidth on each path is 5Mb/s and the latency is 50ms. It is observed that for TCP application, when more than 6 paths are in use, the performance of multipath without double buffer get worse than that of single path connection. As analyzed before, this is due to the persistent reordering problem in TCP. Therefore, persistent reordering problem has serious impact on the TCP performance.

Figure 4.2a also shows that for TCP, PSMC with double buffer can aggregate the bandwidth more effectively. The aggregate bandwidth increases when the number of paths increases. However when there are more than 8 - 9 paths, the aggregate bandwidth actually starts to decline slowly. This is due to multipath overhead. As the number of paths increases, the packet loss rate increases. And the double buffer size gets bigger and takes longer to respond to a real packet loss. All these factors slow down the system performance. As analyzed below, the processing overhead of our PSMC code in Linux kernel is limited and is not the major source of multipath overhead.

In practice, we usually pick 4 to 8 paths to achieve maximum aggregate bandwidth for TCP applications. The bandwidth gain over 10 paths is limited.

Figure 4.2a also shows that for UDP, PSMC without double buffer can effectively aggregate the available bandwidth (Due to the space limitation, we don’t show the UDP bandwidth result after 10 paths). This is because we don’t have to consider problems like persistent reordering and congestion control for UDP.

Figure 4.2b further shows the bandwidth utilization between the four types of connections. The x axis is the number of paths in use. The y axis is the bandwidth utilization, which is the ratio of measured throughput to the total bandwidth available. It is not surprise that the single path connection and the multipath connection without double buffer (TCP) can not utilize the available network bandwidth effectively. For the multipath connection with double buffer (TCP) approach, as the number of paths increases, the bandwidth utilization decreases slowly. For multipath without double buffer (UDP), it can effectively use all the available bandwidth.

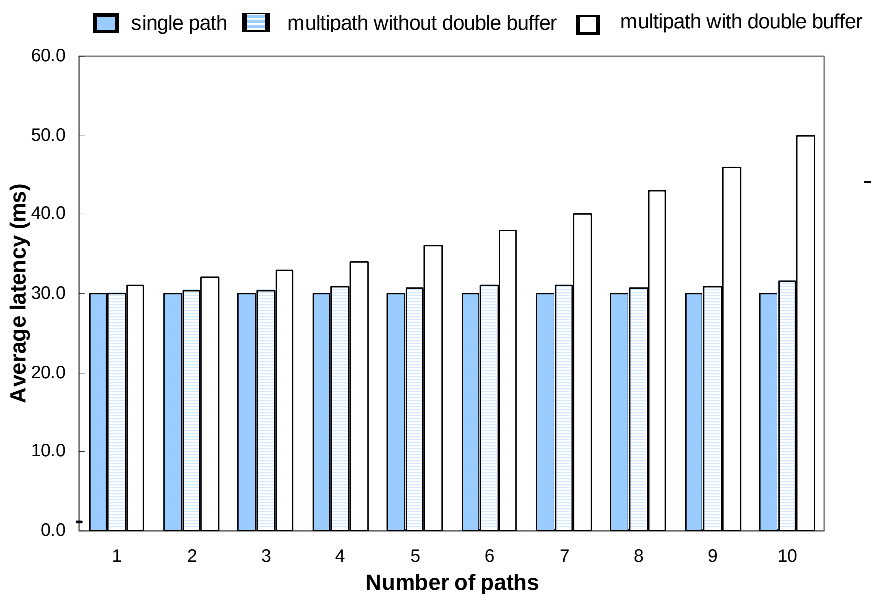


Figure 4.3: PSMC latency analysis

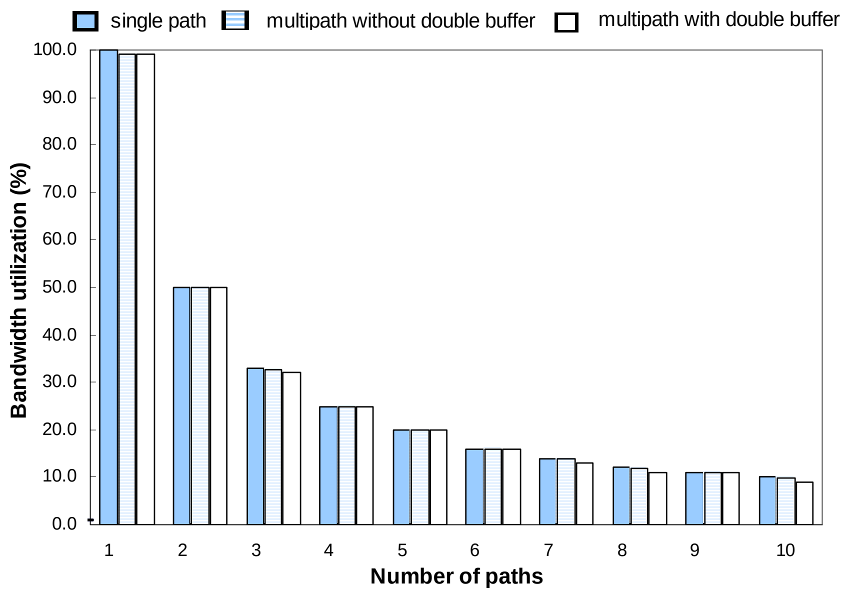


Figure 4.4: processing overhead of PSMC on a single path

b) PSMC performance analysis on latency

Figure 4.3 shows the latency comparison on PSMC. We don’t use the Round Trip Time (RTT) or One Way Delay (OWD) as the latency metrics because they can not properly indicate the latency impact of PSMC. Instead we adopt the sender-perceived response time which is the time period between sending out a packet and receiving the corresponding ACK. We use the average sender-perceived response time over a long period (1000 seconds) during a stable transmission stage as the latency metric. The bandwidth on each path is 5Mb/s and the latency is 30ms. Figure 4.4 indicates that the latency in PSMC with double buffer (TCP) increases when the number of paths increases. This can be explained as follows: when there are more paths involved, the double buffer size increases, the packets are hold in the buffer longer. At the same time, it takes longer to detect a real packet loss. All these factors attribute to a longer average latency.

Figure 4.3 indicates that in a PSMC double buffer system (TCP), if latency is an important factor, then we should not select more than 7 paths. Second, when possible, select less number of paths because it can reduce the latency.

Even though the latency increases in a PSMC double buffer system for the TCP applications, we argue that with the abundant bandwidth available, we can design and utilize some transmission schemes which contain redundant or error correction information to reduce the latency [TNgu03, ABan96].

Figure 4.3 also shows that the latency of PSMC without double buffer keeps almost the same as the single path connection when the number of paths increases. This feather is very helpful for the UDP related applications.

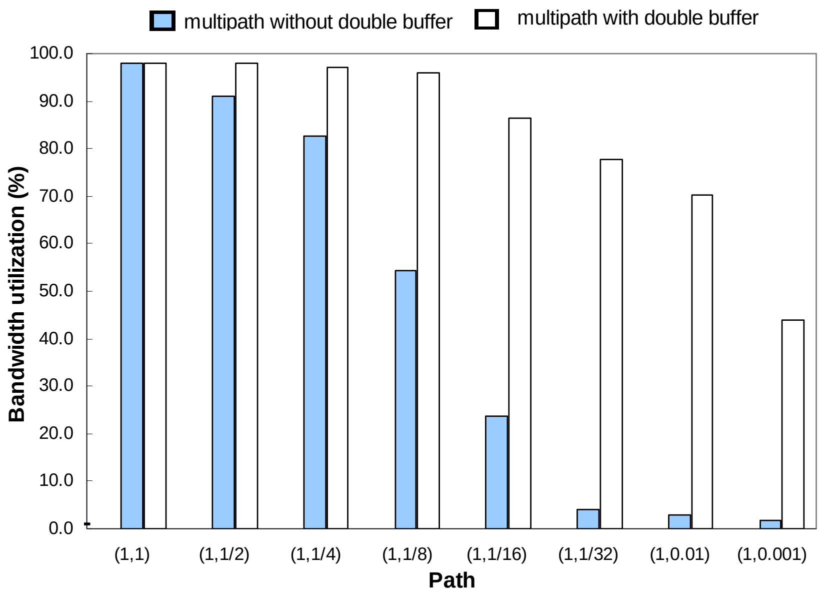


Figure 4.5a: the impact of bad path

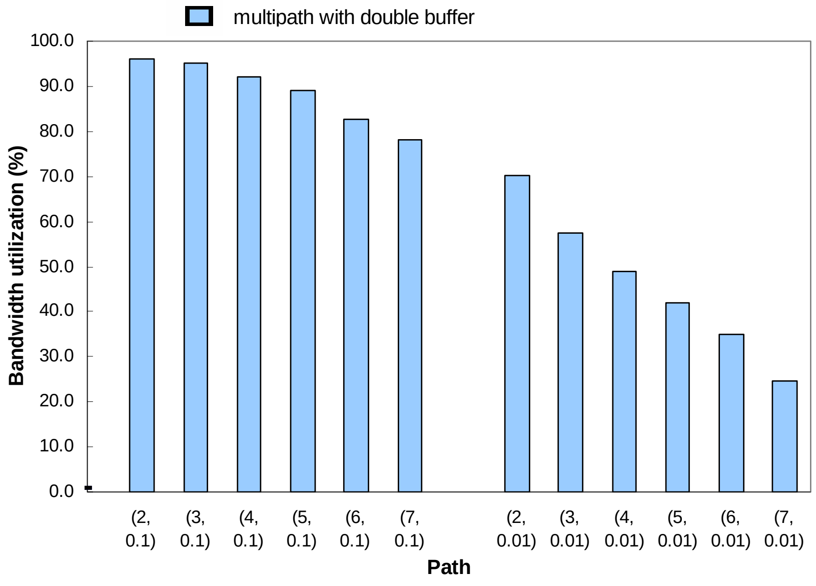


Figure 4.5b: the impact of bad path

c)

Processing overhead of PSMC implementation

From Figure 4.2a, we can also observe that if there is only one path in use, the performance of the four connections is almost the same. This indicates that the processing overhead of PSMC code on the single path may be very limited.

Figure 4.4 further shows that the processing overhead of PSMC over single path is limited. The dark bar is for the single path connection, and the other two bars are for PSMC with only one path in use. It shows that when PSMC is used as single path connection, its performance is comparable with a true single path connection. This indicates that the processing overhead of our PSMC implementation on throughput is limited.

From Figure 4.3, we can also observe that the latency of PSMC is primarily caused by the packet holding time in the double buffer. And the PSMC processing overhead has limited impact on latency.

d) Impact of “bad” path

Figure 4.5a shows the impact of “bad” path with uneven bandwidth distribution. The x axis notation (1, 1/2) means two paths are in use, the bandwidth ratio is 1:1/2. It is observed that the bad or unbalanced paths have big impacts on performance. This is because when the paths become more uneven, the double buffer size gets bigger. When packets are lost, it takes longer to enter fast retransmit mode. Figure 4.4a also shows that the bad path has bigger impact on PSMC without double buffer than with double buffer, because PSMC without double buffer has a poorer ability to deal with packet reordering.

Figure 4.5b further depicts the impact of bad path. The x axis notation (3, 0.1) means three paths are in use, and the bandwidth ratio is 1:1:0.1. We can observe that as the number of paths increases, the impact of bad path gets more significant. This can be explained that the increased number of paths can complicate the TCP reordering problem.

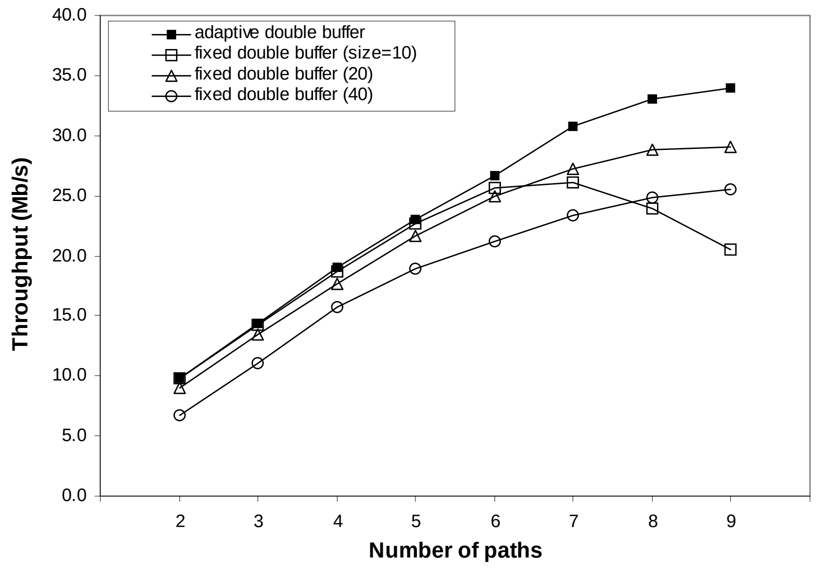


Figure 4.6a: the impact of double buffer scheme

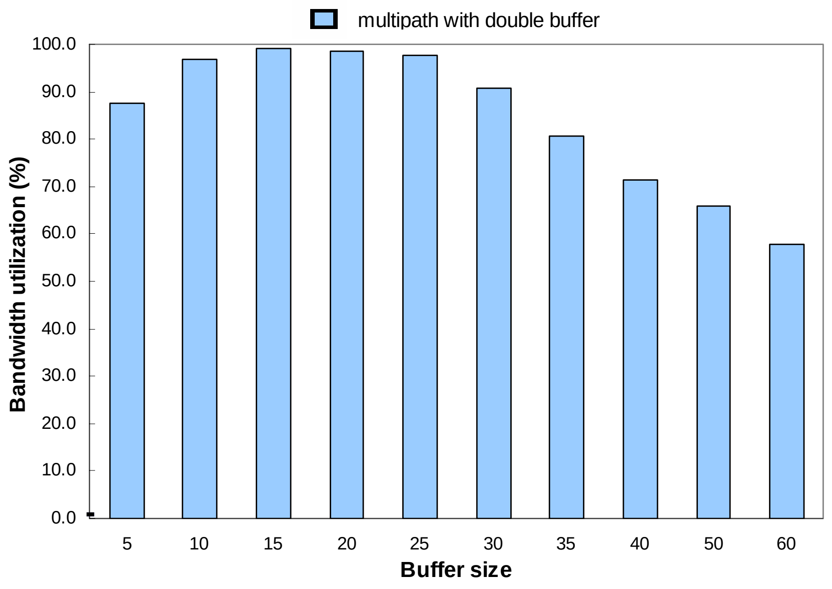


Figure 4.6b: the impact of double buffer size

Figure 4.5a and 4.4.5b suggests that we should eliminate the bad path or uneven path for whose bandwidth is below the 1/10 of the average. Keeping such uneven links in a multipath connection may slow down the performance.

e) Impact of double buffer scheme and new cwnd scheme

Figure 4.6a shows the impact of the double buffer schemes. In the test, the bandwidth on each path is 5Mb/s and the latency is 30ms. The sampling period is 60 seconds. The a parameter is 2. We perform a long http download task in the test. It is observed that the adaptive scheme consistently out-perform the fixed-size schemes. If the buffer size is set to be small (10), it performs well when there are less number of paths. But when the number of paths increases, its performance drops dramatically. This can be explained as follows. When there are less number of paths (less than 6), the buffer size 10 is still OK to temporarily hold the incoming packets. When there are more paths (more than 6), then the buffer size 10 is too small to hold incoming packets, therefore, a lot of unnecessary Dup ACKs are sent out, and the system performance drop dramatically. On the other hand, if the buffer size is set to be too big (40), then it takes too long to respond to a real packet loss. This will significantly degrade the system performance. It is observed that

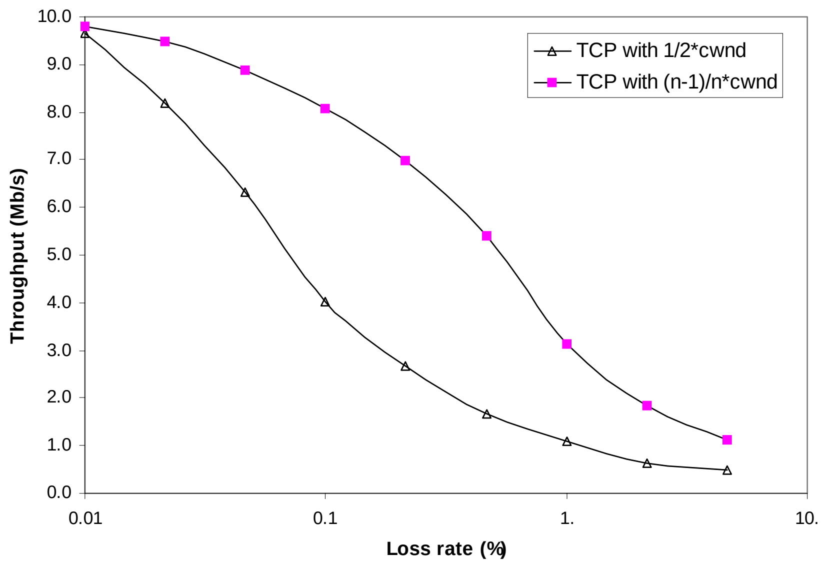


Figure 4.7: the impact of new cwnd scheme

the curve of size 40 performs worst. The buffer size 20 is in between and the performance is in between either.

From Figure 4.6a, we also observe that when there are less than 6 paths in use, the difference between adaptive scheme and fixed size scheme (size 10 and 20) are limited. This suggests that the system performance seems not very sensitive to the double buffer size. In Figure 4.6b we perform another test to see the impact of double buffer size by using fixed-size scheme. It confirms that the PSMC performance is not very sensitive to the double buffer size when the buffer size is in a certain range (10 – 25 in this test). When the buffer size exceeds this range, the performance starts to decline gradually. This is certainly a good news to our double buffer algorithm, since it means an approximate

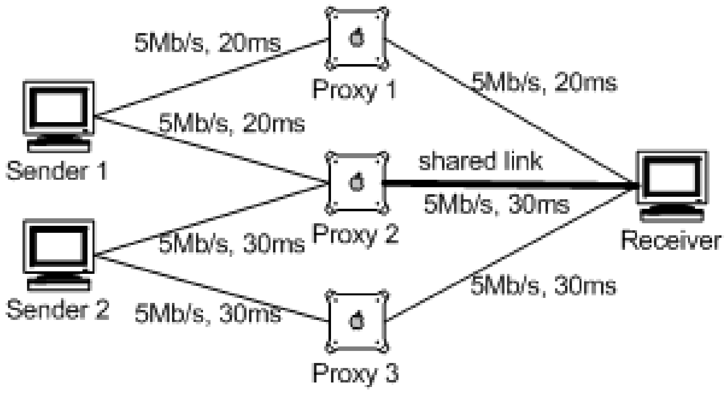


Figure 4.8: test bed for TCP fairness

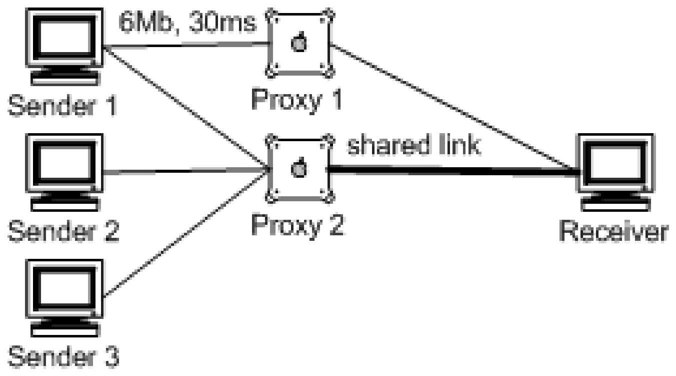


Figure 4.9: test bed for TCP friendliness

update on the double buffer size is acceptable.

Figure 4.7 shows the impact of new congestion window size adjustment scheme by setting 1/2*cwnd to (n-1)/n*cwnd. The x axis is the lost rate in log scale. There are five paths in the test and the bandwidth on each path is 2Mb/s. We run packet dropper on proxy servers to simulate the packet loss. The packet dropping rate can be specified through the /proc file system. As we can see, the new cwnd scheme is consistently out-performs the old cwnd scheme. When the lost rate increases, the throughputs of both schemes decline. However, the new scheme declines slower than old scheme. The effect gets significant in a lossy network environment where packet loss rate is relatively high.

f) PSMC TCP fairness

TCP fairness implies that all connections are provided with similar opportunity to transfer data, and no connection suffers from “starvation”. For example, if K TCP sessions share same bottleneck link of bandwidth C, each has average rate of C/K.

Table 4.2: TCP fairness

TCP flow	Throughput (Mb/s)
sender 1 - proxy 1 - receiver	5
sender 1 - proxy 2 - receiver	2.49
sender 2 - proxy 2 - receiver	2.49
sender 2 - proxy 3 - receiver	5

We run an experiment to evaluate the PSMC TCP fairness. The test bed is shown in Figure 4.8. Sender 1 sends out packets to receiver via proxy 1 and proxy 2. Sender 2 sends out packets via proxy 2 and 3. The shared congestion link is between proxy 2 and receiver. Table 4.2 shows the experimental results. It is observed that the TCP flow can fairly share the bandwidth of the common link (2.49 + 2.49 ≈ 5.0).

g) PSMC TCP friendliness

TCP friendliness implies that PSMC must be "friendly" to other TCP variants. They must be able to coexist with each other while providing opportunities for all connections to progress satisfactorily.

We run an experiment to evaluate the TCP friendliness between PSMC and TCP Reno, which is one of the most commonly used TCP variants. The test bed is shown in Figure 4.9. All links are 6Mb with 30ms latency. Sender 1 sends out packets to receiver via proxy 1 and proxy 2 using PSMC. Sender 2 sends out packets via proxy 2 using TCP Reno as a single path connection. Same for sender 3. The shared congestion link is between proxy 2 and receiver. Table 4.3 is the experimental results.

Table 4.3: TCP friendliness

TCP flow	Throughput (Mb/s)
sender 1 - proxy 1 - receiver	6
sender 1 - proxy 2 - receiver	1.99
sender 2 - proxy 2 - receiver	1.99
sender 3 - proxy 2 - receiver	1.99

h) Path related issues.

Now we study the path related issues. Table 4.4 shows the initial set up time of multiple paths. The delay primarily comes from the secure communication overhead between the participating hosts. The relatively long initial set up time makes PSMC more suitable for long-lived flow. To reduce the path initialization time, paths can be set up ahead of time (before run time). Also, these paths can be shared by different user sessions. We can also observe that the initial set up time is scalable to the number of paths.

Table 4.4: Initial set up time of multiple paths

Number of paths	Set up time (second)
2	10.3
5	12.1
10	14.8
50	20.4

Table 4.5: Path detection, deletion and addition

Action	Time to finish (second)	Attacked flow (second)
Detection	2.1	240 + 12 = 252
Delete	5.8	
Add	5.7	

Table 4.5 shows how long it takes to detect a bad path and how long to delete and add a path dynamically. We first start a web downloading task of 240 seconds by using 5 paths in parallel. During the downloading process, we launch a DDoS attack against a selected path. It takes 2.1 seconds to detect the "bad" path (not through IDS, but through PSMC passive bandwidth monitoring). Then the sender deletes the bad path and recruits a new path (of the same bandwidth). It takes about 5.8 seconds to finish such action. During the test, the traffic flow continues without stopping. With the deletion and addition action, the traffic flow finishes in 252 seconds. The overhead of deletion/addition action is about (252-240)/240 = 5%, within an acceptable range for long-lived flow.

i) UDP tests

Table 4.6a: UDP test (2Mb/s paths only)

Number of paths	1	2	3	4	…	10
Aggregate bandwidth (Mb/s)	2	4	6	8	…	20

Table 4.6b: UDP test (one 200Kb/s, the rest 2Mb/s paths)

Number of paths	1	2	3	4	…	10
Aggregate bandwidth (Mb/s)	0.2	2.2	4.2	6.2	…	18.1

Table 4.6c: UDP test (one 20Kb/s, the rest 2Mb/s paths)

Number of paths	1	2	3	4	…	10
Aggregate bandwidth (Mb/s)	0.02	2.02	4.02	6.01	…	18.01

We also run several UDP tests on PSMC. The first test uses real player video streaming. We play a constant-bit-rate (CBR) video at a rate of 5Mb/s. There are 10 paths of 2Mb/s available. By using a single path connection, the video constantly pauses and enters the buffering mode. By using 3 paths in parallel (2M * 3 = 6M > 5M), the video can be viewed smoothly.

The second UDP test uses a UDP packet generator which can generate a large amount of UDP packets. There are 10 paths of 2Mb/s, 1 path of 200Kb/s and 1 path of 20Kb/s available. Table 4.6(a-c) shows the result of UDP tests.

It is observed from the above tests that for UDP, PSMC can effectively utilize the aggregate bandwidth, and the bad path has very limited impact on the aggregate bandwidth.

We also run a UDP/TCP competition test. There is a multipath connection of two sub-paths with total bandwidth of 6Mb/s, and 2 TCP flows of 2Mb/s each running on the connection. Then we launch a large UDP flow without rate control. We observe that the UDP flow quickly consumes most of the available bandwidth, leaving little share for TCP flows. Then we enforce a rate limiting on UDP packets (1Mb/s) at sender side, and the TCP flows start to recover. Table 4.7 illustrates the test result.

Table 4.7: UDP and TCP competition test (Mb/s)

	TCP1	TCP2
Before UDP starts	2	2
After UDP starts	0.1	0.1
UDP with rate control	2	2

Conclusion

In this chapter, we propose the design and implementation of the Proxy Server based Multipath Connection (PSMC), which can set up multiple routes between two end hosts and utilize them in parallel by striping packets across these routes. We summarize the key issues in a multipath system and provide solutions in PSMC. The experimental results show that PSMC can make good usage of network resources and significantly improve the network performance, security and reliability.

CHAPTER V

PROXY SERVER SELECTION ALGORITHMS

Proxy server selection or path selection is a critical decision in a multipath connection network. Different selection may result in significantly different result. In this chapter, we present several algorithms including genetic algorithms to solve the proxy server selection problem in multipath connection environment.

Introduction

In the previous two chapters, we present a new multipath connection approach called Proxy Server based Multipath Connection (PSMC), in which the multiple paths are set up via a set of intermediate connection relay proxy servers. The proxy servers examine the incoming packets and forward them to the appropriate destination.

One of the key issues in a multipath system is path selection, or proxy server selection. We will mix the usage of path selection and proxy server selection in this chapter. There might be a large number of proxy servers available; we need to select the "optimal" subset of proxy servers from the candidates and achieve the objective functions, like maximum aggregate bandwidth. Different path selection may result in significantly different performance [SSav99, RON01]. Therefore, server selection is a critical decision in PSMC.

When the network paths are disjointed, the network reliability is improved, the available throughput increases, the traffic along the paths are load-balanced and least likely to be correlated. Therefore, finding multiple disjoint paths are usually desirable in multipath environment.

There are two types of disjoint: link disjoint if no common links between paths, and node disjoint if no common nodes between paths besides the end host nodes. In general a link-disjoint paths algorithm can be extended to a node-disjoint algorithm with node splitting [GYFK03, JSRT84].

Network Model

By using networking measurement tools, like traceroute [Trac], pathchar [Path], cprobe [Cpro], we can obtain the IPs of the routers along the selected path and estimate the bandwidth or latency on each link in that path. Extensive work has been done in network measurement [SSav99, VPax, SJCJ00]. Usually we can get the network topology between given end hosts based on the network measurement techniques. Figure 5.7 is an example network topology from a node at UCCS network lab to the selected Redhat mirror servers.

For simplicity, in this chapter we assume the network topology is known and static. Even though it is not always true [VPax], we argue that in a short given period this assumption is acceptable. We also assume the network bandwidth on network links is known and static. Due to the dynamic nature of Internet, the available bandwidth on network links is not a constant and keeps changing. However, with more accurate bandwidth probing approaches, it is possible to measure the network bandwidth in a short period. We also assume the network forwarding route and return route are same. Even though it is not always true either [VPax], we argue that the results in this chapter on an undirected graph can be extended to a directed graph.

We model the proxy server selection problem as followings.

Let $G = (V, E)$ be a graph which models the network topology. V represents the set of nodes including proxy server nodes, end host nodes, and the routers in between; E represents the set of edges or link segments that connect the nodes in the network.

Let P be the set of proxy servers, $P = \{p_1, p_2,\ldots p_n\}$. The source node is s and destination is d. For a proxy server p_i, $BW(p_i)$ represents the available bandwidth of the indirect route via p_i from s to d, and $L(p_i)$ represents the latency of the indirect route via p_i. For an edge e, BW(e) represents the available bandwidth on edge e, and L(e) represents the latency on edge e. Assume the route via p_i consists of a set of edges e_1, e_2 …, e_m, we define

$$BW(p_i) = \min\{BW(e_1), BW(e_2)\ldots BW(e_m)\},$$

and

$$L(p_i) = L(e_1) + L(e_2) + \ldots + L(e_m).$$

Let S be the subset of proxy servers that we selected, $S \subseteq P$.

The proxy server selection problem is to find a subset S from P to maximize the objective functions and meet some constraints in graph G.

NP Hardness

There are many open problems in server selection problem or sever placement problem. Problems like mirror server and cache server placement and selection problems, are topics gaining interests recent years [EYYM, LQVP01, SJCJ00, PKDR00, BLMG99]. In this chapter, we concentrate on selecting disjoint paths, which has very different objective functions with the cache server problem. For a related work survey on disjoint paths, please refer to chapter 2.

We summarize the disjoint path selection problem in multipath environment and study their NP-hardness as below.

1) Max-aggregate bandwidth problem.

The problem of finding maximum aggregate bandwidth in a given network is close to the classic maximum flow problem [CPKS82] and can be solved in polynomial time. The basic idea is to use augmenting path and labeling scheme.

Instance: A graph G = (V, E), with capacity $BW_{i,j}$ associated with each edge $e_{i,j}$, a source node s, a destination node d,

Question: Is there a set of paths S = {p_1, p_2, … p_k} from s to d such that $\sum_{p_i \in S} BW(p_i)$ is maximum?

Lemma 5.1: The Max-aggregate bandwidth problem can be solved in polynomial time.

Prove: The max-aggregate bandwidth can be easily converted to the classic maximum flow problem, therefore it can be solved in polynomial time. The complexity of max-aggregate bandwidth problem is the same as the maximum flow problem. We denote the complexity as O(c(maxflow)).

2) Max-bandwidth, k-disjoint path problem.

This problem refers to the problem of finding k disjoint path whose aggregate bandwidth is maximum. This problem can be converted to the well-known K-best paths problem in [SLCW99]. The K-best paths problem is to find k disjoint paths which have the lowest total cost. It can be solved in polynomial time. The basic idea is to use node splitting to transfer the K-best path problem into the classic maximum flow minimum cost problem.

Instance: A graph G = (V, E), with capacity $BW_{i,j}$ associated with each edge $e_{i,j}$, a source node s, a destination node d, K is a given positive integer.

Question: Is there a set of node-disjoint paths S = {p_1, p_2, … p_k} (set size is K) from s to d such that $\sum_{p_i \in S} BW(p_i)$ is maximum.

Theorem 5.1: The max-bandwidth, k-disjoint path problem can be solved in polynomial time.

Proof:

The K-best paths problem is as follows: Given a graph G_2 = (V, E), with capacity $BW_{i,j}$ and cost $C_{i,j}$ associated with each edge $e_{i,j}$, a source node s, a destination node d, K is a given positive integer. Let $C(path_m) = \sum_{e_{i,j} \in path_m} c_{i,j}$. Find a set of node-disjoint paths {p_1, p_2, … p_k} (set size is K) from s to d such that $\sum_{i=1}^{k} C(P_i)$ is minimum.

We can convert the max-bandwidth, k-disjoint path problem to the K-best paths problem. For the given graph G = (V, E) in max-bandwidth, k-disjoint path problem, we convert it to graph G_2 by assigning zero cost to all the edges. The edge $e_{i,j}$ capacity in G_2 is $BW_{i,j}$, same as in G. Define a "virtual" edge from the origin s to the destination d with a flow cost of one unit and no upper or lower bound on capacity. This virtual edge is node-disjoint with all other edges. See Figure 5.1. We assign a large number as the "supply quantity" at the origin s, and same quantity is "demanded" at the destination d.

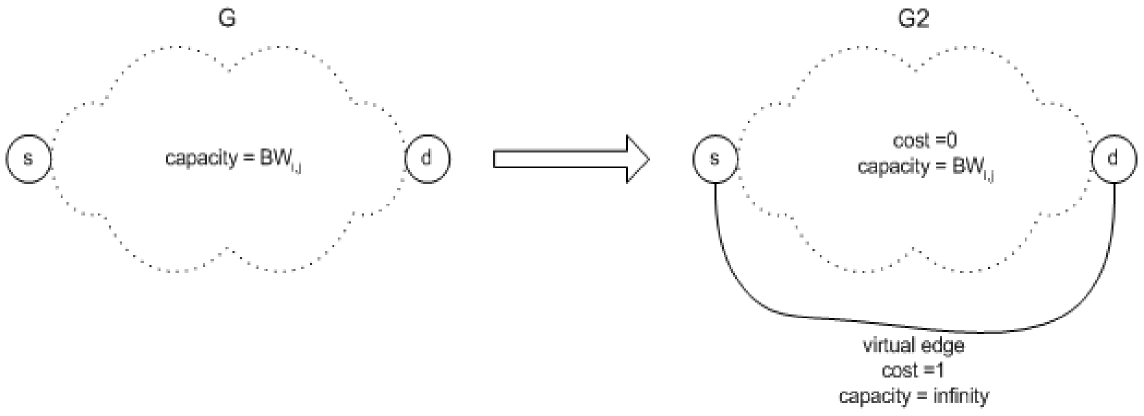


Figure 5.1: conversion from G to G2

In trying to find the minimum cost flow with k-best paths algorithm in G_2, the solution procedure will push the maximum possible flow through the original network since it costs nothing on these edges. Only the overflow over and above the maximal flow possible in the original network is diverted through the virtual edge from the origin s to the destination d. The maximum aggregate bandwidth in G is of total supply flow amount minus virtual edge flow amount. Therefore, the solution in G_2 by using k+1 best paths algorithm also yield the solution in G with k disjoint path of maximum aggregate bandwidth.

As we can see from the proof, the complexity of the max-bandwidth, k-disjoint path problem is the same as the K-best paths problem, which is also the same as the well-known Maximum flow and minimum cost network flow problem (MCNF). The best known algorithm for MCNF is

$$O(c(mcnf)) = O(c(n,m,k)) = O(\min\{A1,A2,A3\}),$$

where $A1=nm\log(n^2/m)\log nM$, $A2=nm(\log\log k)\log(nM)$ and $A3=(m\log n)(m+n\mathrm{longn})$. Here c(n,m,k) is the complexity of MCNF problem, n is the number of nodes in G, m is the number of links in G, M is a big number.

3) Max-bandwidth, min-number-of-disjoint-paths problem.

This problem refers to finding a set of disjoint paths, whose aggregate bandwidth is maximum, and the set size (number of paths) is minimum. This problem is important because in practice people want to use least number of disjoint paths to achieve the maximum aggregate bandwidth. As we can see from chapter 4, the more number of paths are used, the less efficient of bandwidth utilization, and the longer latency. When exceeding a threshold (8-10), adding paths will not provide additional bandwidth.

Instance: A graph G = (V, E), with capacity $BW_{i,j}$ associated with each edge $e_{i,j}$, a source node s, a destination node d.

Question: Is there a set of node-disjoint paths S = {p_1, p_2, … p_k} (set size is K) from s to d such that $\sum_{p_i \in S} BW(p_i)$ is maximum and k is minimum.

Lemma 5.2: The maximum number of link-disjoint paths in G can be found in polynomial time.

Proof: For the given graph G = (V, E), we convert it to graph G_1 by assigning unit capacity to all edges. See Figure 5.2.

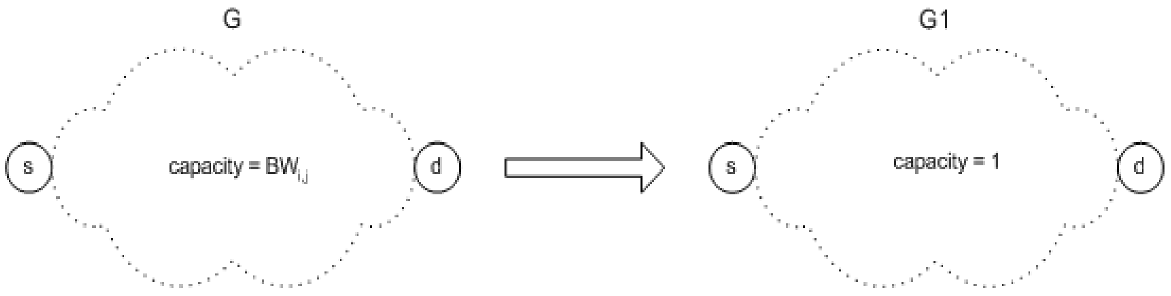


Figure 5.2: G to G1

Then we apply maximum flow algorithm on graph G1. Assuming the maximum flow is K, since the link capacity is one unit, so there must be K paths in the solution path set. Assume the solution path set S = {p_1, p_2, … p_k}. First, p_1, p_2, … p_k must be link disjoint, otherwise the maximum flow is less than K. Second, there is no more link disjoint path in G1, otherwise the maximum flow is more than K. Therefore K is the maximum number of disjoint paths in G1 and G, and it can be found in polynomial time.

.

Lemma 5.3: The maximum number of node-disjoint paths in G can be found in polynomial time.

Proof: For the given graph G = (V, E), we first convert it to a directed graph G1 by duplicating edges and orienting them both ways [RATM93]. We then convert G1 to graph G2 by splitting nodes (except s and d). As illustrated in Figure 5.3, we convert G to G_1 by duplicating edges, then split node i in G_1 to node m and n in G2 by adding a virtual link between m and n with the capacity of one unit. All the incoming edges go to one node and outgoing edges go to the other. The capacities of all other links in G2 are also set to be one unit.

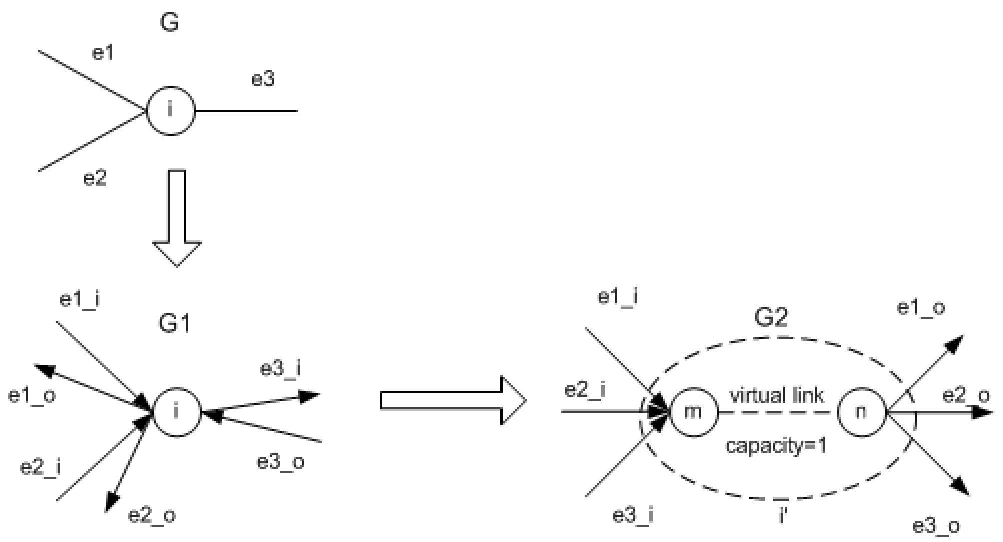


Figure 5.3: conversion from G to G_1 to G_2

We then apply the max-flow algorithm on graph G_2. Assuming the maximum flow is K. From lemma 5.2, we know that the maximum number of link-disjoint paths in G_2 is K. Assume the solution path set S = $\{p_1, p_2, \ldots p_k\}$.

Translated back from G_2 to G_1, the link-disjoint paths selected in G_2 are node-disjoint paths in G_1. Then translated back again from G_1 to G, note that the duplicated edges in G_1 can not be selected into the solution set at the same time. For example, e_{1_i} and e_{1_o} can not be both in the solution set in G_1. Therefore we can translate the solution set in G_1 back to G. Then we get the maximum number of node-disjoint paths in G, and it can be obtained in polynomial time.

Theorem 5.2: The max-bandwidth, min-number-of-disjoint-paths problem can be solved in polynomial time.

Prove: We first use lemma 5.1 to find the maximum aggregate bandwidth in G, assuming is W. Then we use lemma 5.2, 5.3 to find the maximum number of link disjoint paths L and node disjoint path N, assuming the aggregate bandwidth is WL and WN respectively.

If W > WL, then it is not possible to find link disjoint paths to achieve maximum aggregate bandwidth. If W = WL, then we can achieve maximum bandwidth W with minimum number of link disjoint paths L.

Same for node disjoint. If W > WN, then it is not possible to find node disjoint paths to achieve maximum aggregate bandwidth. If W = WN, then we can achieve maximum bandwidth W with minimum number of node disjoint paths L.

Easy to see, the complexity of the proposed algorithm is the same as the max-flow algorithm, which is denoted as O(c(maxflow)).

4) Max-bandwidth, min-longest-latency path problem.

In multipath environment, path latency is also an important factor. It is critical for some real time multimedia applications. When selecting multiple paths, people usually want to find a set of disjoint paths to achieve maximum aggregate bandwidth, at the same time, the path with longest latency in this set is minimum among all possible path sets.

Instance: A graph G = (V, E), with capacity $BW_{i,j}$ and latency $L_{i,j}$ associated with each edge $e_{i,j}$, a source node s, a destination node d, let path latency $L(P) = \sum_{i,j \in p} L_{i,j}$.

Question: Is there a set of node-disjoint paths S={p_1, p_2, … p_k} from s to d such that $\sum_{p_i \in S} BW(p_i)$ is maximum, and the maximum path latency $L(P_m)$ (m=1…k) is minimum. Or given a non-negative number X, is the latency of the slowest path $L(P_m)$ is less than or equal to X?

Theorem 5.3: The max-bandwidth, min-longest-latency path problem is NP-Complete.

Proof: We show the problem is NP-complete by giving a transformation from a well known NP-complete problem – the Partition problem [MGDJ79].

The partition problem: given a finite set A and a size s $s(a) \in Z+$ for each $a \in A$, is there a subset $A' \subseteq A$ such that

$$\sum_{a \in A'} s(a) = \sum_{a \in A-A'} s(a)$$

It is easy to see that this problem is NP, since a non-deterministic algorithm can get a set of disjoint paths with maximum aggregate bandwidth and check if the length of the longer path is less than or equal to X.

Let’s construct an instance of the problem with a graph G = (V, E), with the following characteristic:

1) (3n + 2) nodes, s is source node, d is destination node

2) $V = \{v_i, i=1\ldots2n\} \cup \{u_i, i=1\ldots n\} \cup \{s, d\}$

3) The edges are:

a) (s, v_1), (s, u_1), (v_{2n}, d), (u_n, d)

b) (v_i, v_{i+1}), i=1…2n-1

c) (u_i, u_{i+1}), i=1…n-1

d) (v_{2i}, u_{i+1}), i=1…n-1

e) (u_i, v_{2i+1}), i=1…n-1

4) The bandwidth on all edges $BW_{i,j} = 1$

5) The latency is as follows:

a) $L(v_{2i-1}, v_{2i}) = s(a_i)$, i=1…n

b) The Latency of every other edge is 0

(0 means a very small number compared with $s(a_i)$)

6) The $X = 1/2 * \sum_{ai \in A} s(ai)$

It is easy to see that there are at most two disjoint paths from s to d, and the maximum aggregate bandwidth is 2. We need to show that instance of the original problem will have two disjoint paths from s to d of length at most X, if and only if elements of the instance of the Partition problem can be divided into two groups, such that the sums of these two groups are equal.

First, suppose that the set A can be divided into A’ and A-A’ such that $\sum_{a \in A'} s(a) = \sum_{a \in A-A'} s(a)$. In this case we need to show that we can construct two disjoint paths from s to d that the longest latency is at most $1/2 * \sum_{ai \in A} s(ai)$. Suppose A'={$a_{f(1)}$, $a_{f(2)}$… $a_{f(m)}$}, f(1) < f(2)<…<f(m).

The two paths can be constructed as follows.

Path 1: $s \rightarrow u_1 \rightarrow \dots \rightarrow u_{f(1)-1} \rightarrow v_{2f(1)-1} \rightarrow v_{2f(1)} \rightarrow u_{f(1)+1} \rightarrow \dots \rightarrow u_{f(2)-1} \rightarrow v_{2f(2)-1} \rightarrow v_{2f(2)} \rightarrow u_{f(2)+1} \rightarrow \dots \rightarrow u_{f(m)-1} \rightarrow v_{2f(m)-1} \rightarrow v_{2f(m)} \rightarrow u_{f(m)+1} \rightarrow \dots \rightarrow u_n \rightarrow d$.

Path 2: $s \rightarrow v_1 \rightarrow \dots \rightarrow v_{2f(1)-2} \rightarrow u_{f(1)} \rightarrow v_{2f(1)+1} \rightarrow v_{2f(1)+2} \rightarrow \dots \rightarrow v_{2f(2)-2} \rightarrow u_{f(2)} \rightarrow v_{2f(2)+1} \rightarrow v_{2f(2)+2} \rightarrow \dots \rightarrow v_{2f(m)-2} \rightarrow u_{f(m)} \rightarrow v_{2f(m)+1} \rightarrow v_{2f(m)+2} \rightarrow \dots \rightarrow v_{2n} \rightarrow d$.

We can verify that the path latency is X and the aggregate bandwidth is 2.

Second, now suppose that there are two disjoint path P1 and P2 from s to d, the latency of slower path P2 is at most X and the aggregate bandwidth is 2.

It is easy to see that for edge (v_{2i-1}, v_{2i}) with latency s(a_i), i=1…n, it must be part of P1 or P2, and can not be in P1 and P2 at the same time. So L(P1) + L(P2) =X. We also have L(P1) <= L(P2) and L(P2) <=1/2*X, therefore, L(P1) = L(P2) = 1/2*X.

Assuming P1 contains edges with latency of $a_{f(1)}$, $a_{f(2)}$… $a_{f(m)}$, f(1) < f(2)<…<f(m), then we can set A'={$a_{f(1)}$, $a_{f(2)}$… $a_{f(m)}$}, and the sums of two groups A' and A-A' are equal 1/2*X. Therefore, we prove the theorem.

5) Max-bandwidth, min-jointness problem.

In the real world scenarios, there may not exists disjoint paths between two given end hosts, because the paths are likely to share some common links on the edge of the Internet. Another scenario is as follows: two end hosts are in China and US respectively, since there are limited gateways connections between China and International network

[IChi], therefore, the multiple paths between China and US are likely to share some common links in the middle.

We define jointness and disjoint function as follows like [MZha04].

Jointness = (total number of shared links) / (total number of links)

Disjoint = 1 - (total number of shared links) / (total number of links) = 1 - jointness

We can also use bandwidth or latency instead of the number of links in the definition.

We study the problem of selecting a set of paths to achieve maximum aggregate bandwidth with minimum jointness.

Instance: A graph G = (V, E), with capacity $BW_{i,j}$ associated with each edge $e_{i,j}$, a source node s, a destination node d.

Question: Is there a set of paths S = {p_1, p_2, … p_k} from s to d such that $\sum_{p_i \in S} BW(p_i)$ is maximum, and S is as diverse as possible? ($\sum_{p_{i,j} \in S} Jointness(i, j)$ is minimum among all possible path set who have maximum aggregate bandwidth).

Another objective function is to maximize $\sum_{p_{i,j} \in S} (\alpha * BW(p_i) + Disjoint(i, j))$, here α is a given parameter. This objective function combines both bandwidth and jointness.

The max-bandwidth, min-jointness problem appears to be NP-complete. We propose a heuristic algorithm as follows.

In Theorem 5.2, when W > WL, then it is not possible to find link disjoint paths to achieve maximum aggregate bandwidth. Only if W = WL, then we can achieve maximum bandwidth W with minimum number of link disjoint paths L.

When W > WL, we first apply Theorem 5.2 to find a set of link disjoint paths S = {p_1, p_2, … p_k} which bandwidth is WL. Then we convert graph G to G_1 by assigning one unit cost on the selected path $\{p_1, p_2, \ldots p_k\}$, and zero cost on all other edges. We also need to deduct the used bandwidth on each edge in $\{p_1, p_2, \ldots p_k\}$. Then we convert graph G_1 to G_2 by removing the edges with zero bandwidth. Figure 5.6 is an example that illustrates the above conversion.

Then we apply maximum flow minimum cost algorithm on G_2. Assume the solution path set S' = $\{q_1, q_2, \ldots q_m\}$. Easy to see, the maximum aggregate bandwidth in G_2 is (W-WL). Since set S has unit cost, therefore the S' use as much zero cost edges as possible. This means S' has as little common links with S as possible. However, note that S' itself may not be a disjoint set. Therefore, by combing S and S', we can get a heuristic solution which can achieve maximum aggregate bandwidth W and are diverse.

6) Constraints

There are also some constraints on the path selection. The experimental results in PSMC show that the bandwidth distribution among the selected multiple paths has significant impact on the overall system performance [YCai05]. A large-capacity link and a small-capacity link may have worse aggregate bandwidth than two moderate links. The experimental results suggest that if the bandwidth of a path is smaller than 1/10 of the average of the bandwidth, then this path is treated as “bad” path and should be eliminated.

Another constrain is the number of paths selected. The experimental results in PSMC show that the total number of paths should be smaller than a threshold (usually 7-10) [YCai05]. The bandwidth gain over 10 paths is limited, or even become negative.

7) Parallel download from multiple mirror sites

With the recent development of internet, we are able to retrieve documents from multiple server sites, like the mirror sites, to increase the downloading speed, make better use of available network bandwidth and parallel processing speed of servers. Recent work by Rodriguez, Kirpal, and Biersack [RKB00] studied how to use the existing HTTP protocol for retrieving documents from mirror sites in parallel to reduce the download time and to improve the reliability. The proposed approach utilizes the HTTP 1.1 byte range header to retrieve specific data in a mirror server site, which requires no changes on existing server and client settings.

The algorithms we study for proxy server selection problem can be applied to the parallel download from multiple mirror sites problem.

Heuristic Path Selection Algorithms

With the objective functions and constraints, some path selection problems in a multipath system are NP-Complete. Heuristic algorithm is a feasible solution. In this chapter we propose to use genetic algorithm and a greedy algorithm to solve the problems.

The three example objective functions are:

a) Maximize $\sum_{p_i \in S} BW(p_i)$ and maximize $\sum_{p_{i,j} \in S} Disjoint(i, j)$

b) Maximize $\sum_{p_{i,j} \in S} (\alpha * BW(p_i) + Disjoint(i, j))$, here α is a given parameter.

c) Maximize $\sum_{p_i \in S} BW(p_i)$, and minimize $\max\{p_i \in S \mid L(p_i)\}$.

The two example constraints are:

a) $BW(p_i) > T * \sum_{p_i \in S} BW(p_i) / Sizeof(S)$, here T is a parameter set to be 1/10.

b) $Sizeof\ (S) < W$, here W is a parameter set to be 10.

1) Genetic Algorithm

We proposed to use genetic algorithm to solve the NP-complete problem in path selection. The reasons why to choose genetic algorithm are as follows.

a) It provides more flexibility and extensibility on this problem. If the objective functions and constraints are later changed, we can easily modify the fitness function in genetic algorithm to accommodate such changes. Other heuristic algorithms may require more significant modification under such circumstances.

b) It provides better scalability. The execution time of genetic algorithm scales well with regard to the network size.

c) It provides more controls for the end users. It can easily produce multiple outputs and give end users the opportunity for further selection.

The disadvantages of genetic algorithm are as follows.

a) It is a heuristic algorithm and can not always give the optimal answer.

b) The execution time might be long for a small scale network.

We implement a fix-length genetic algorithm in which the length of chromosomes is fixed, and a variable-length genetic algorithm in which the length of chromosomes can change.

The genetic algorithm works as below.

1) Assign sequential server number, node number and path number to denote each proxy server, node and path. Assign the initial bandwidth to each path.

2) Initialize the first generation of chromosomes by filling server number in chromosome. For better performance, we put the last known best results into the first generation.

3) Crossover and mutation at certain probability. Make sure no duplicated server in chromosome, and the length of chromosome is less than the given upper limit. Several different crossover and mutation methods have been combined together for better performance [JKoz92].

4) Calculate fitness function. For a given chromosome, use the objective function as fitness function, and check constraints.

5) Run certain generations, and output the stabilized result.

2) Greedy Algorithm

For the maximum bandwidth minimum jointness problem, we also proposed a heuristic algorithm based on Section 5.3-5 to solve the path selection problem as follows.

1) Initialize the data set.

2) First use lemma 5.1 to find the maximum aggregate bandwidth, assuming is W. Then use lemma 5.2, 5.3 to find the maximum number of link disjoint paths L, assuming the aggregate bandwidth is WL. We have W > WL. Assume the solution path set S = {p_1, p_2, … p_k}.

3) Convert the original graph G to G_2 by assigning unit cost on set S and deducting set S from G, as illustrated in Figure 5.6.

4) Apply maximum flow minimum cost algorithm on G_2. Assume the solution path set S' = $\{q_1, q_2, \ldots q_m\}$.

5) By combing S and S', we can get a heuristic solution which can achieve maximum aggregate bandwidth W and are diverse.

This is a greedy algorithm. As we can see from the algorithm, the execution time will be polynomial. Every time when we change the objective functions, we need to design new greedy algorithm.

Results Analysis

We tested the proposed algorithms for max-bandwidth min-jointness problem on simulated network topologies as well as a real-world network topology.

GT-ITM [GITM], which is one of the most commonly used internet topology models, is used to generate network topologies of various sizes for evaluating the performance of the proposed algorithms. We randomly pick 10% nodes as proxy servers, and two nodes as end host nodes.

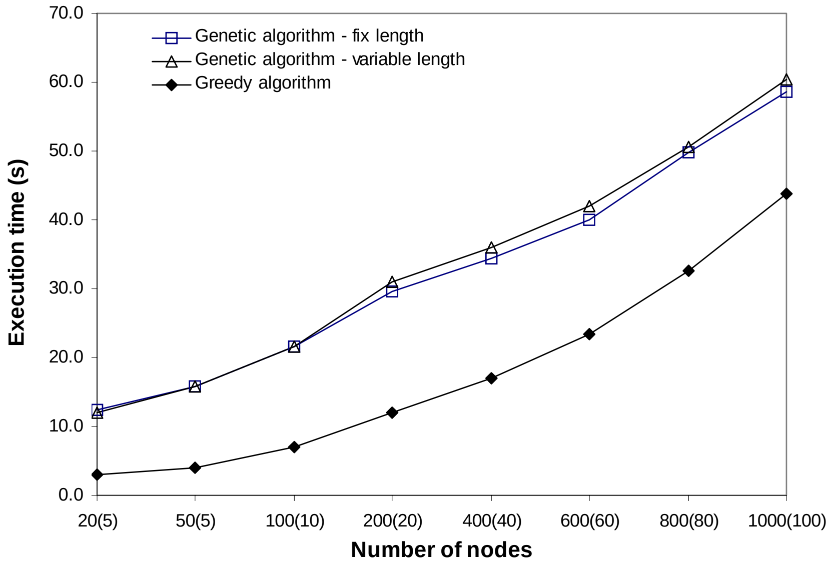


Figure 5.7: algorithm execution time

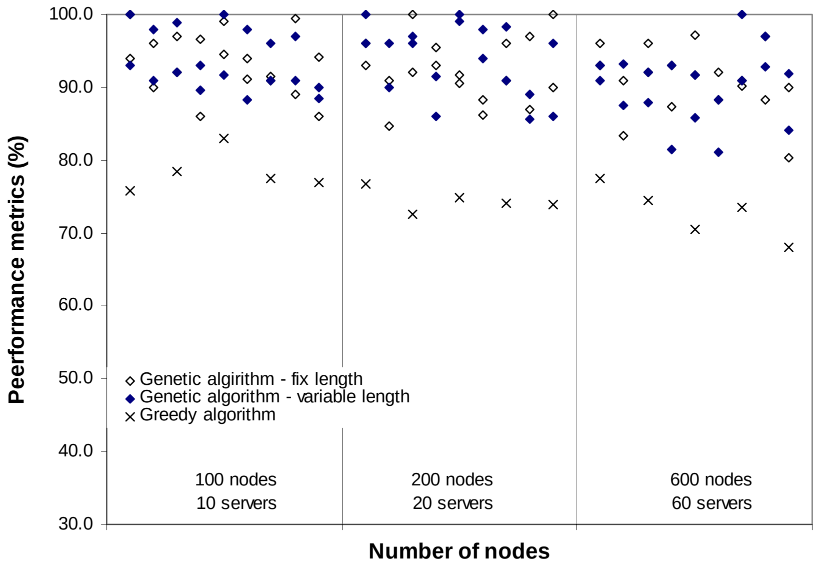


Figure 5.8: algorithm running results

Figure 5.7 shows the algorithm execution time vs. the simulated network size. The x axis notation 20(10) means there are 20 nodes plus 10 proxy server nodes. For genetic algorithm, it usually output stable results after 100 generations. It is observed that the execution time of both algorithms increases when the size of network increases. The greedy algorithm has lower execution time, but as we can see from Figure 5.7, the running result is not satisfactory.

Figure 5.8 shows the distribution of the algorithm running results. The notation on the chart "100 nodes, 10 servers" means there are 100 nodes in the network plus 10 proxy server nodes. We use $\sum_{p_{i,j} \in S} (\alpha * BW(p_i) + Disjoint(i, j))$ as the performance metrics so that we can compare the running results fairly. The parameter α is used to normalize the BW(p). Assuming the maximum aggregate bandwidth is W, then α = 1/W. The disjoint (i, j) is already a normalize value in the range of (0 - 1). For the given network topology, we run each algorithm 25 times. We set the best running result as 100%, and compare other running results against it. For greedy algorithm, we change the number notation in graph G for each run so it can yield different running results.

We can observe from the chart that the genetic algorithm can yield satisfactory result in the range of (80-100) %, which are close to the optimal result (100) %. There is no significant difference between fix-length genetic algorithm and variable-length algorithm. It is also observed that the greedy algorithm can not yield good result (60-80) %. This is because the greedy algorithm tends to yield results with bad disjoint.

Figure 5.9 shows the real network topology from a node at University of Colorado at Colorado Springs (UCCS) to the selected Redhat mirror servers. This topology can be viewed as half of a PSMC network, with Redhat mirror sites selected as proxy server. We use the topology and perform some tests on our algorithm. Table 5.1 shows the running result. It is observed that the execution times of all algorithms are in acceptable range. Both algorithm yield the optimal result because the simplicity of the given network topology.

Table 5.1: running results on a real-world topology

Algorithm	Execution time (s)	Running result / Optimal result
Genetic algorithm – fix length	7.1	100%
Genetic algorithm – variable length	7.2	100%
Greedy algorithm	3.1	100%

Conclusion

Multipath connection is a topic gaining interest. Path selection is a critical decision in a multipath connection network. In this chapter, we present genetic algorithm to solve the path selection problem in a multipath connection environment. Genetic algorithm has better flexibility and extensibility when the context of problem changes. From the performance result, it is observed that genetic algorithm can produce satisfactory results within reasonable execution time.

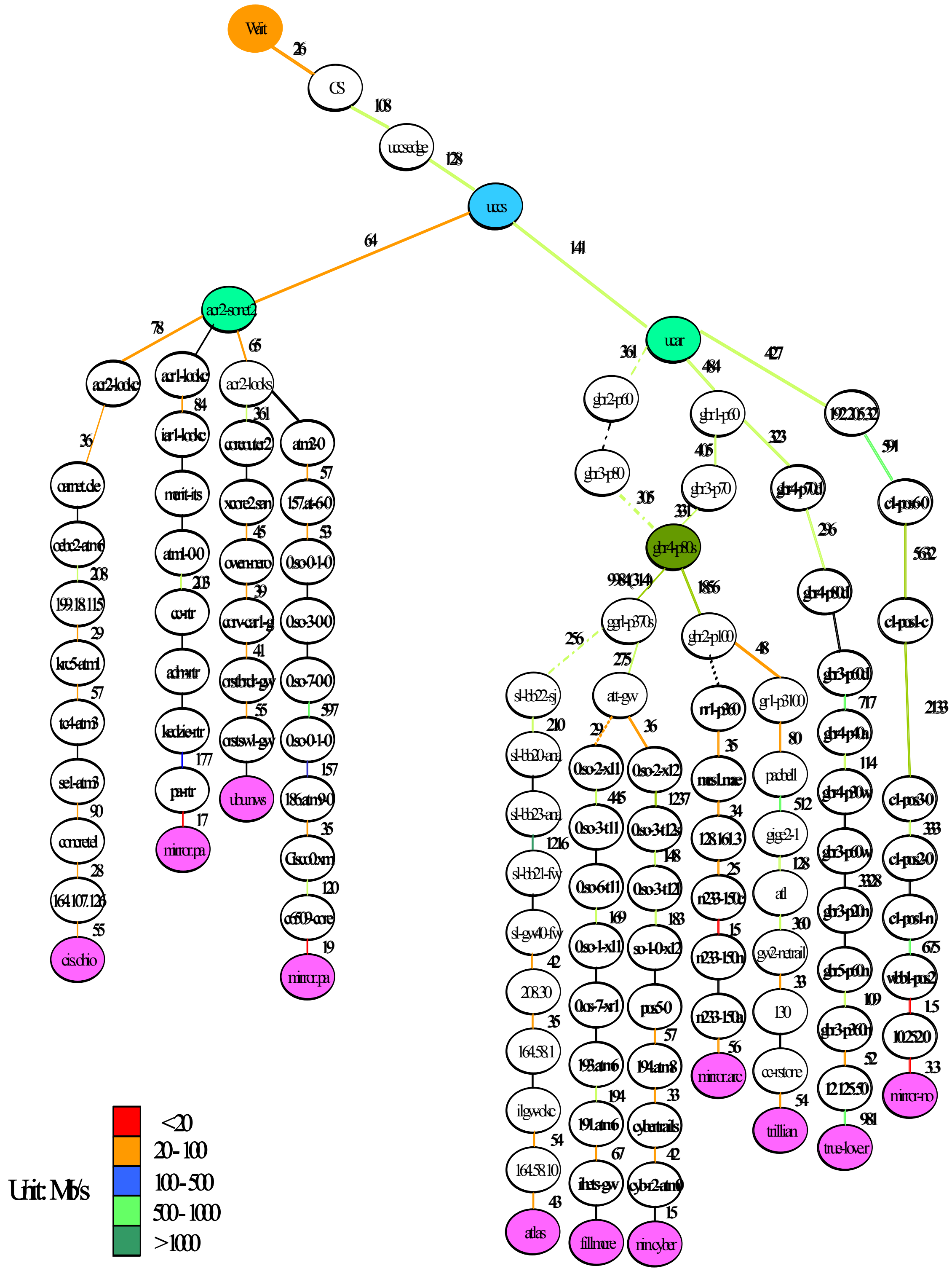


Figure 5.9: network topology from a node at UCCS to the selected Redhat mirror servers [Chow00]

CHAPTER VI

PROPORTIONAL DIFFERENTIATION PROVISIONING

With the abundant bandwidth provided by PSMC, the end server capacity may become the performance bottleneck. There is an increasing demand for provisioning of different levels of quality of services (QoS) on the end server. By combining multipath on network with service differentiation on the end server, we can provide a comprehensive solution for various applications to improve the performance, security and reliability of the overall system.

In this chapter, we present several processing rate allocation schemes based on queueing theory and feedback control theory for proportional service differentiation. We implement the process allocation approaches on an Apache Web server to achieve the processing rates allocated to the request classes.

Introduction

Due to the open and dynamics nature of Internet applications, the last decade has witnessed an increasing demand for provisioning of different levels of quality of service (QoS) to meet changing system configuration and resource availability and to satisfy different client requirements. This differentiated QoS provisioning problem was first formulated by the Internet Engineering Task Force in the network core. Differentiated

Services (DiffServ) [SBDB98] is a major architecture, where the network traffic is divided into a number of classes. It aims to define configurable types of packet forwarding in network core routers, which can provide per-hop differentiated services to per-class aggregates of network traffic. The proportional differentiation model [CDDS99] states that certain class QoS metrics should be proportional to their pre-specified differentiation weights, independent of the class loads. Due to its inherent differentiation predictability and proportionality fairness, the model has been accepted as an important DiffServ model and been applied in the proportional queueing-delay differentiation (PDD) in packet scheduling [CDDS99, CDDS02, MLJL01, BYPM02, JWCX04] and proportional loss differentiation in packet dropping [YHRG04].

There are recent efforts on differentiation provisioning on end servers [TAKS02, JAMD98, SCCE00, XCPM02, HZHT01]. On the server side, response time is a fundamental performance metric. Existing response time differentiation strategies are mostly based on priority scheduling in combination with admission control and content adaptation [TAKS02, JAMD98, SCCE00]. The work in [XCPM02] adopted priority scheduling strategies, strict or adaptive, to achieve response time differentiation on Internet servers. The results showed that the differentiation can be achieved with requests of higher priority classes receiving lower response time than those of lower priority classes. However, this kind of strategies cannot quantitatively control quality spacings, say proportionally, among the classes. Time-dependent priority scheduling algorithms developed for PDD provisioning in packet networks can be tailored for PDD provisioning on Web servers [SLJL04]. However, they are not applicable for proportional response time differentiation because the response time is not only dependent on a job's queueing delay but also on its service time, which varies significantly depending on the requested services. Providing proportional response time differentiation on Web servers is not only important, but also challenging.

Processing Rate Allocation

The proportional responsiveness differentiation model aims to control the ratios of the average responsive time of classes based on their normalized differentiation parameters. Let T_i denote the average response time of requests of class i. Specifically, the model requires that the ratio of average responsive time between class i and j is fixed to the ratio of the corresponding differentiation parameters

$$\frac{T_i}{T_j} = \frac{\delta_i}{\delta_j} \qquad 1 \le i, j \le N \tag{6.1}$$

The differentiation predictability property requires that higher classes receive better service, i.e., lower responsive time. Without loss of generality, we assume that class 1 is the "highest class" and set $\delta_0 < \delta_1 < \delta_2 < \ldots < \delta_N$.

We adopt a M/M/1 FCFS queue for modeling the traffic. Recent Internet workload measurements indicate that for some Web applications a heavy tailed distribution may be more accurate for service time distributions. However, we note that the focus of this Section is on adaptive process allocation for achieving different processing rates in support of responsiveness differentiation. The processing rate allocation scheme derived by an M/M/1 queueing model can give the key insights about the differentiation problem and the feasibility of the process allocation strategy.

We partition the request processing rate of a Web server into N virtual servers. Each virtual server handles requests of one class in a FCFS manner. Let μ_i denote the normalized request processing rate of the virtual server i. We have

$$\sum_{i=1}^{N} \mu_i = 1 \tag{6.2}$$

Assume requests of class i in Poisson process arrive at virtual server i in a rate λ_i. It follows that the traffic intensity on the server $\rho_i = \lambda_i / \mu_i$. According to the foundations of queueing theory, when $\rho_i < 1$, we have the expected response time of requests in class i as

$$T_i = \frac{\rho_i}{\mu_i(1-\rho_i)} = \frac{1}{\mu_i - \lambda_i} \qquad 1 \le i \le N \tag{6.3}$$

For feasible processing rate allocation, we must ensure that the system utilization $\sum_{i=1}^{N} \lambda_i \le 1$. That is, the total processing requirement of the N classes of traffic is less than the Web server's processing capacity. Otherwise, a request's response time can be infinite and responsiveness differentiation would be infeasible. Admission control mechanisms can be applied to drop requests from lower classes so that the constraint holds.

According to the definition of (6.3), the set of (6.1) in combination with (6.2) lead to

$$\mu_i = \lambda_i + \frac{1 - \sum_{i=1}^{N} \lambda_i}{\delta_i \sum_{i=1}^{N} \frac{1}{\delta_i}} \tag{6.4}$$

$$T_i = \frac{\delta_i \sum_{i=1}^{N} \frac{1}{\delta_i}}{C - \sum_{i=1}^{N} \lambda_i} \tag{6.5}$$

From (6.5), we have the following three basic properties regarding the predictability and controllability of the proportional responsiveness differentiation given by the processing rate allocation strategy:

1. Response time of a request class increases with its request arrival rate.

2. With the increase of the differentiation parameter of a request class, its response time increases but all other request classes have lower response times.

3. Increasing the workload (request arrival rate) of a higher request class causes a larger increase in response time of a request class than increasing the workload of a lower request class.

Process Allocation on End Server

A Fixed Process Allocation Strategy

On a process-per-request Web server such as Apache, a process is treated as the scheduling entity for an independent activity. It is also the entity for the allocation of resources, such as CPU cycles and memory space. Process abstraction serves both as a protection domain and as a resource principal. Thus, it is reasonable to assume that the processing rate of a virtual server is proportional to the number of active processes allocated to its process pools.

On an Apache Web server, we can impose an upper bound on the number of processes listening to a port. This maximum number is usually set to be 32 (or 64). To achieve the processing rate ratios between classes, a straightforward solution is to partition 32 processes into multiple process pools listening to different ports. Each pool works as a virtual server handling requests of a class in FCFS manner. Thus, we expect to achieve the processing rates for different classes. We refer to this solution as fixed process allocation strategy since the number of total processes allocated to the pools is fixed.

The problem with the fixed process allocation strategy is that not all allocated processes are always active due to the workload dynamics. For example, we consider a two-class response time differentiation scenario. Given the arrival rates, suppose the calculated processing rate ratio of class 1 to class 2 ($\mu 1$: $\mu 2$) is 3:1. According to the fixed process allocation strategy, 32 processes are partitioned into 24 and 8 and allocated to the process pool of classes 1 and 2, respectively. However, due to the workload dynamics of two classes, it is likely that only 18 of 24 processes of class 1 are active while all 8 processes of class 2 are active. Thus, the real processing rate ratio of class 1 to class 2 is 2:1, instead of 3:1. The fixed process allocation strategy may not be able to achieve proportional response time differentiation. We are going to show its results later.

A Queueing - theoretical Process Allocation

We propose a queueing-theoretical adaptive process allocation strategy. Its objective is to dynamically and adaptively change the number of processes allocated to process pools for handling different classes while ensuring the ratio of process allocations specified by the queueing-theoretical processing rate allocation scheme. The rationale is that, to achieve the processing rate ratios among classes, the allocation strategy has to assure that most of the processes allocated to the process pools listening to corresponding ports are active. To utilize the advantage of the Apache pre-forking mechanism, it allows a small number of processes on a port to be idle. The number is identified by a threshold (H). If more than H processes on a port are idle, the approach is to decrease the number of processes allocated to all process pools proportionally.

Algorithm 6.1 gives the details of the approach.

Algorithm 6.1

A queueing-theoretical adaptive process allocation approach.

```
1: for each process allocation period do
2: get the number of active processes ($p_i$) currently allocated to port i from Apache scoreboard; let P = $\sum_{i=1}^{N} p_i$ ;
// Apache server automatically forks new processes according to the workload condition
3: get the normalized process allocations $\mu_1, \mu_2 \ldots \mu_N$ according to (6.4);
4: search for a process multiplier m, so that $m\sum_{i=1}^{N}\mu_i \le P < (m+1)\sum_{i=1}^{N}\mu_i$
//$m\mu_i$ is the number of processes that the allocation strategy wants to allocate to port i.
// $p_i$ is the number of active processes on port i, which is adjusted in the following.
5: for each port number i do
6:     while $p_i - m\mu_i > H$ // too many processes forked on port i
7:             prohibit a process on this port from listening new requests;
               // this process will soon become idle and be killed by Apache itself.
8:     end for
9: end for
```

In each process allocation period, a multiplier m is used to keep the ratio of the number of active processes of process pools to the normalized value specified by the allocation scheme (6.4). At line 3, the normalized process allocation (μ_i) is the normalized integer value of the number of processes allocated to the process pool i. For example, in a two-class scenario, if μ1/μ2 = 3/1, we have μ1 =3 and μ2 =1. At line 4, a desirable value of m is searched. It is incremented if the total number of active processes of all process pools (P) is greater than the target total number ($m\sum_{i=1}^{N}\mu_i$). This scenario is possible due to the pre-forking mechanism of Apache Web servers. For instance, although the allocation strategy initially assigns 3 and 1 processes for listening port 1 (process pool for handling class 1) and port 2 (process pool for handling class 2), respectively, the Apache server may actually have forked 10 and 4 processes for listening the two ports respectively. Line 7 adjusts the allocations to ensure the ratio of process allocations among the classes. It lets Apache itself to prohibit a process from listening new requests.

An Integrated Process Allocation Approach with Feedback Control

To provide fine-grained proportional response time differentiation, we propose to design a feedback controller and integrate it with the queueing-theoretical adaptive process allocation approach. Proportional integral derivative (PID) control is one of the most classical control design techniques widely used in industrial control systems [GFJP02]. In our system, PID controller is used to adjust the number of processes allocated to a process pool according to the difference between the target average response time and the experienced average response time of a request class. Specifically, the operation of the PID controller is described as follows:

$$p_i(k+1) = p_i(0) + K_P e_i(k) + K_I \sum_{j=0}^{k-1} e_i(j) + K_D \Delta e_i(k) \qquad (6.7)$$

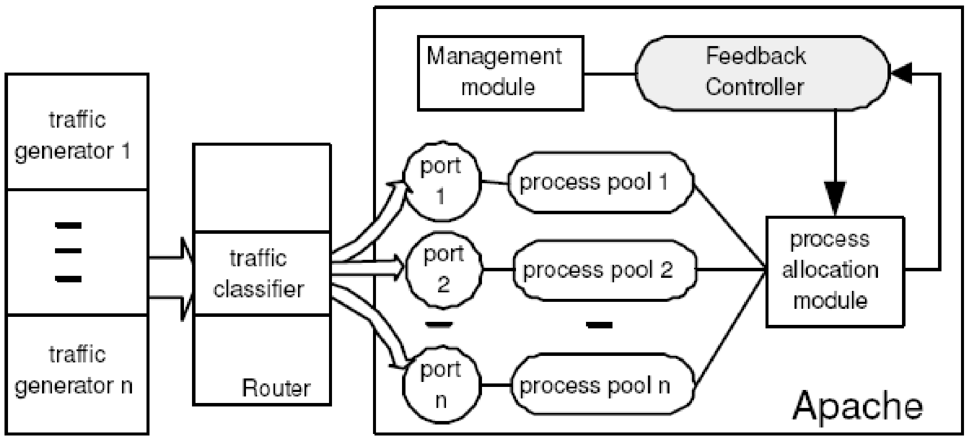


Figure 6.1: The implementation structure

$p_i(0)$ denotes the initial number of processes allocated to process pool i according to the queueing-theoretical process allocation approach. The other three terms added to $p_i(0)$ in the equation above denote proportional, integral, and derivative components, respectively. Setting a larger proportional feedback gain (K_P) typically leads to faster response at the cost of increasing system instability. The derivative control takes into account the change of errors ($\Delta e_i(k)$) in adjusting the process allocation of a class and hence responds fast to errors. Increasing the derivative gain (K_D) typically results in higher system stability.

Performance Evaluation

Implementation Issues

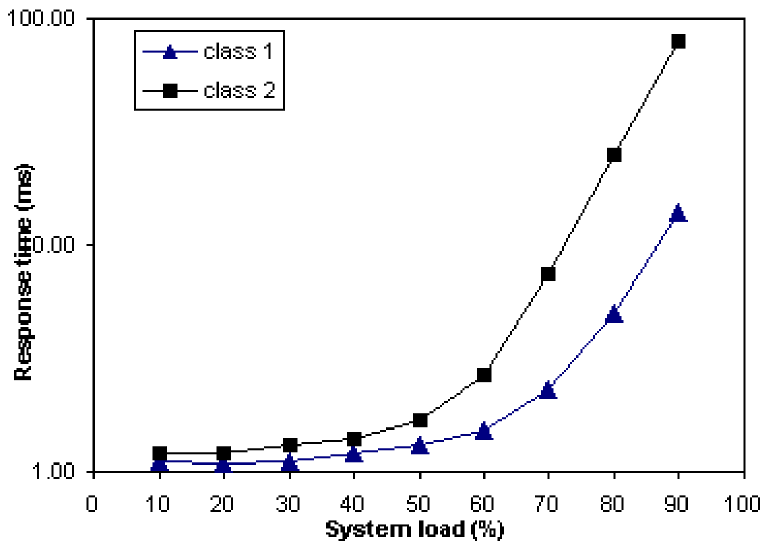
We implemented the process allocation strategies on an Apache Web server to evaluate the impact of the feedback control on the proportional response time differentiation. Figure 6.1 depicts the architecture of the integrated process allocation implementation. Two HP PCs (PIII 1 GHz, 516M RAM) installed with Redhat 9 were used as a router and a Web server, respectively. Four HP PCs (PIII 233 MHz, 96MB RAM) installed with Redhat 9 and Httperf 0.8 [DMTJ] were used to generate Http requests of Poisson distribution. The router conducted traffic classifications. We installed Apache 1.3.29 on the Web server. We configured Apache server at application level to make one server listen to different ports. The number of ports was determined by the number of traffic classes to be differentiated. The requests of class 1 were routed to port 80 which was handled by the process pool 1, requests of class 2 were routed to port 8000 handled by the process pool 2, and requests of class 3 were routed to port 8080 handled by the process pool 3.

The process allocation module in the Web server calculated the processing rate of each class according to its predicted load condition. The load was predicted for every sampling period, which was the processing time of one thousand average-size requests. We adopted a moving window with exponential averaging for the load prediction. The predicted load was the average of past five sampling periods. We implemented the process allocation approaches by modifying child main() function in http main.c file of the Apache server. The process forking and killing mechanisms were not modified and still handled by Apache. This application-level implementation is flexible and portable.

Performance Evaluation

a) Fixed process allocation

Figure 6.2(a) shows the achieved average response time of class 1 and 2 under various system load conditions. The arrival rate ratio of two classes is 3:1. The differentiation weight ratio is set to be 1:3. The fixed process allocation strategy dynamically partitions all 32 processes into the two process pools for class 1 and class 2 according to their changing arrival rates. It can be seen that requests of class 1 always receive lower response time than those of class 2. This demonstrates that the responsive time differentiation is achieved by the processing rate allocation scheme.



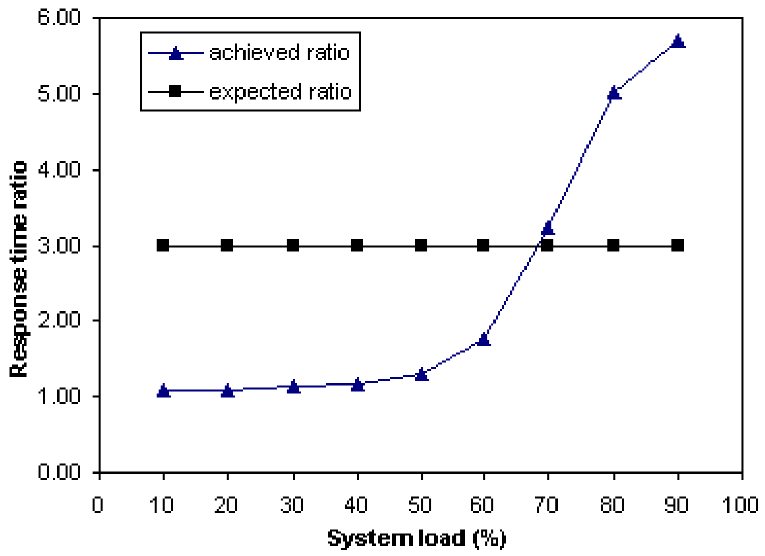
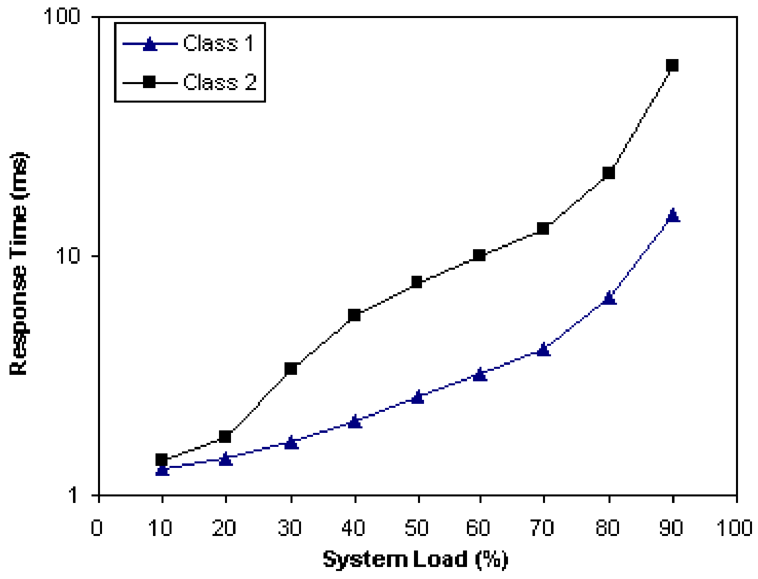


Figure 6.2(a-b): Achieved average response time and response time ratio for fix process allocation

Figure 6.2(b) shows the achieved response time ratio of class 2 to class 1. The achieved ratio is very different from target ratio. This can be explained by the fact that the variance of interarrival time distributions and the variance of service time distributions affect the performance of process allocation and scheduling significantly.

The fixed process allocation strategy cannot achieve proportional response time



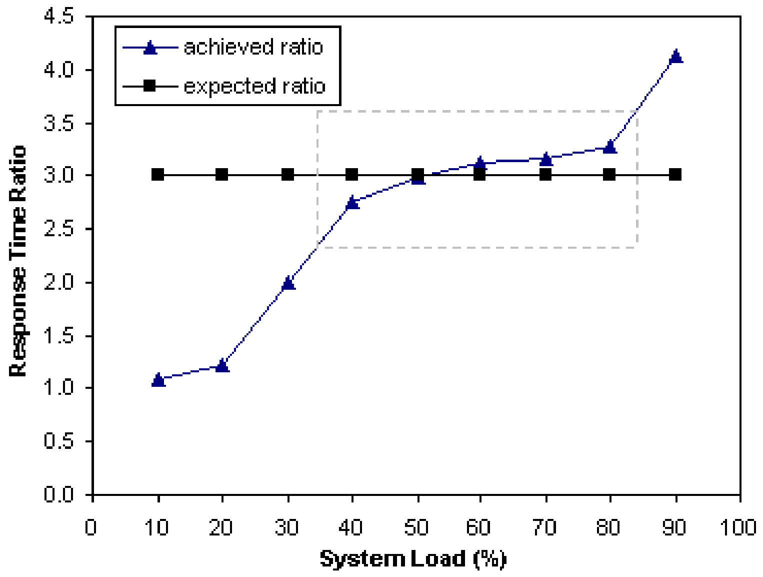
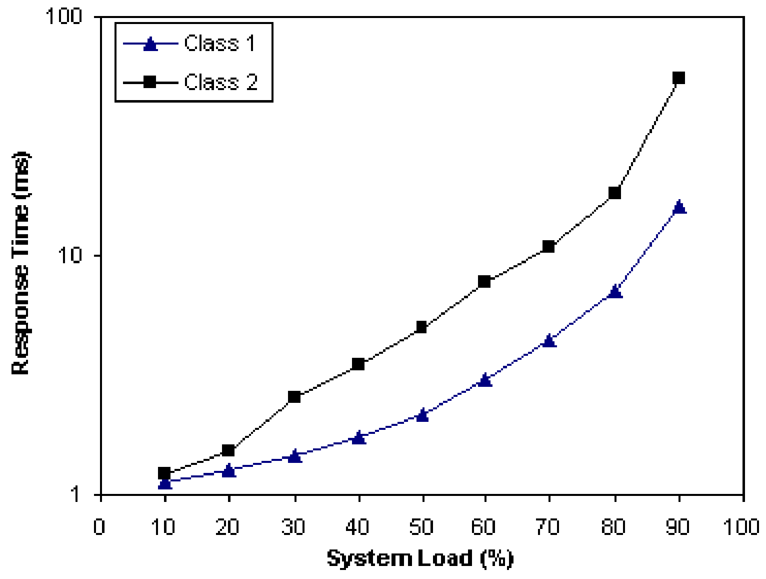


Figure 6.3(a-b): Achieved average response time and response time ratio for adaptive queueing-theoretical process allocation (δ_1: δ_2=1:3)

differentiation because the processing rate of classes cannot be achieved accurately due to the workload dynamics.

b) Adaptive queueing-theoretical Process Allocation

Figure 6.3(a) depicts the achieved response time of classes 1 and 2 due to the adaptive



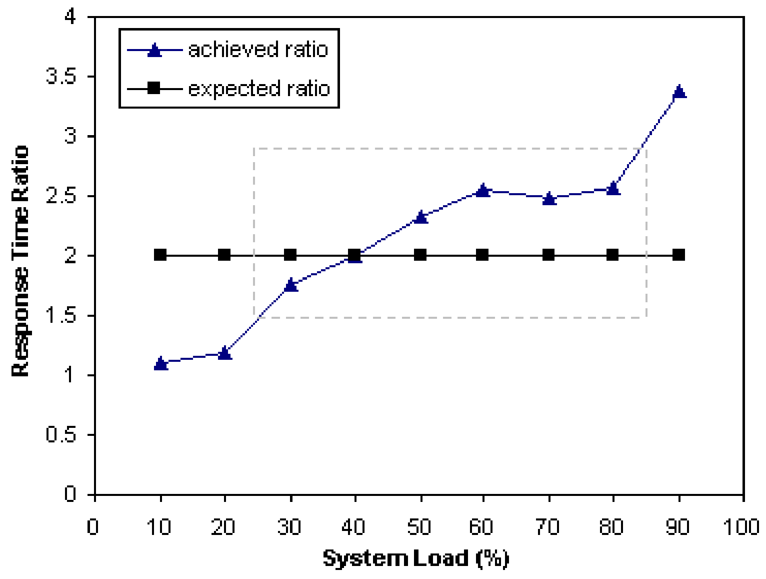


Figure 6.4(a-b): Achieved average response time and response time ratio for adaptive queueing-theoretical process allocation (δ_1: δ_2=1:2)

process allocation strategy under various system load conditions. The arrival rate ratio of two classes is 3:1. The differentiation weight ratio of two classes is 1:3. It shows that the adaptive allocation strategy can achieve response time differentiation. That is, requests of class 1 always receive lower response time than requests of class 2.

Figure 6.3(b) further depicts the achieved response time ratio of class 2 to class 1. When the system load is between 40% to 80%, we can see that the proportional response time differentiation can be achieved. The difference between the achieved response time ratio and the expected ratio is trivial.

As we know, process abstraction serves both as a protection domain and as a resource principal in current general purpose operating systems. However, because an application has no control over the consumption of resources that the kernel consumes on behalf of the application, resource principals do not always coincide with processes. We believe that this problem is one of the primary reasons for the difference between the achieved ratio and the expected ratio. There is a demand for new kernel-level resource management mechanisms, such as resource container, a new operating system abstraction introduced recently. Figure 6.3(b) also shows that when the arrival rate is below 30%, the expected response time ratio is not achieved. This is explained by the fact that when the workload is light, there is almost no queueing delay observed in all traffic queues. Note that the request scheduling policy is work conserving. Therefore, DiffServ is not feasible under certain light load conditions, as it was also observed in experiments for PDD provisioning in packet networks. When the system load is higher than 90%, we also find out that the expected ratio is not achieved. This can be explained that as the system load is close to its capacity, the impact of the variance of incoming traffic on queueing delay dominates and thus queueing delay in all traffic queues increases significantly. This affects the controllability of the process allocation strategy significantly.

Figure 6.4(a) depicts the achieved response time of classes 1 and 2 due to the adaptive process allocation strategy under various system load conditions. We change the differentiation weight ratio of two classes from 1:3 to 1:2. The arrival rate ratio of two

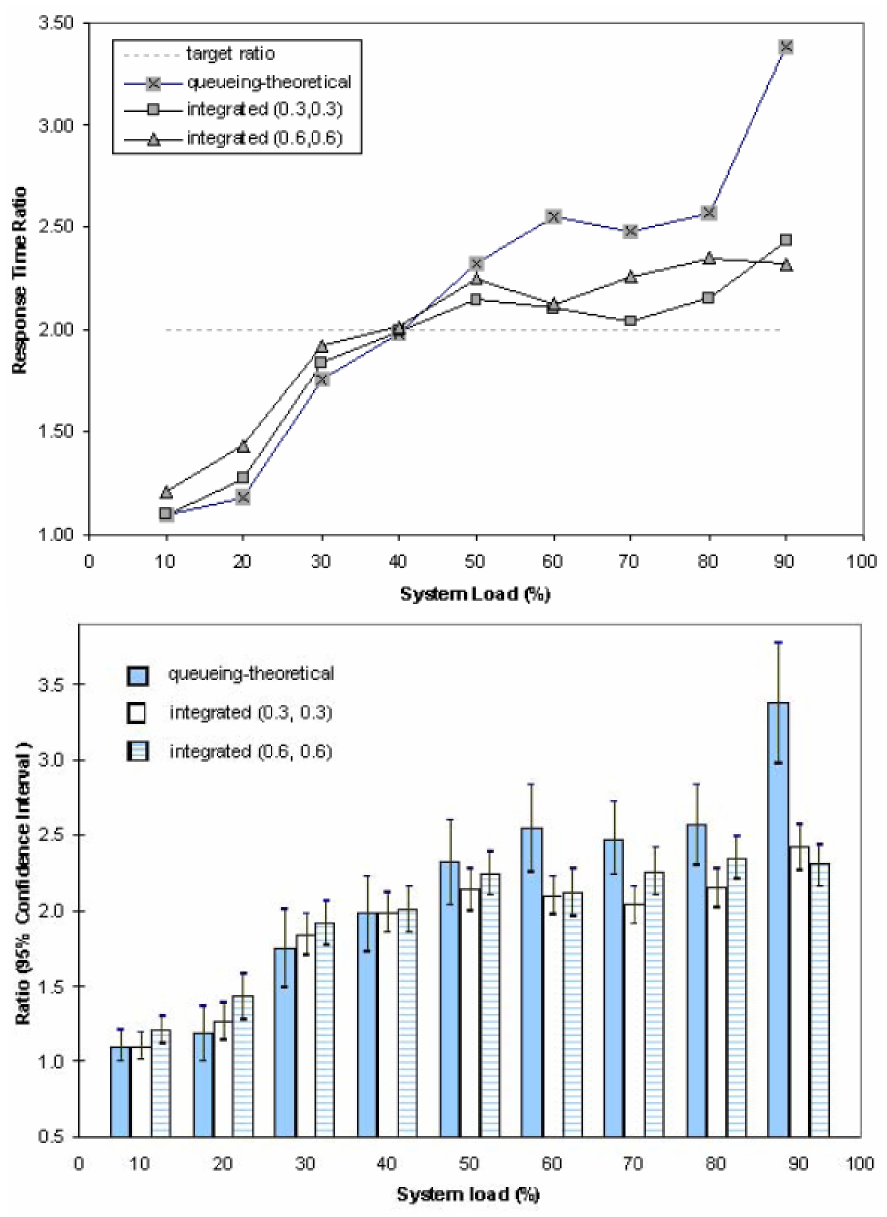


Figure 6.5(a-b): Achieved average response time ratio and 95% confidence intervals for integrated process allocation (δ_1: δ_2=1:2)

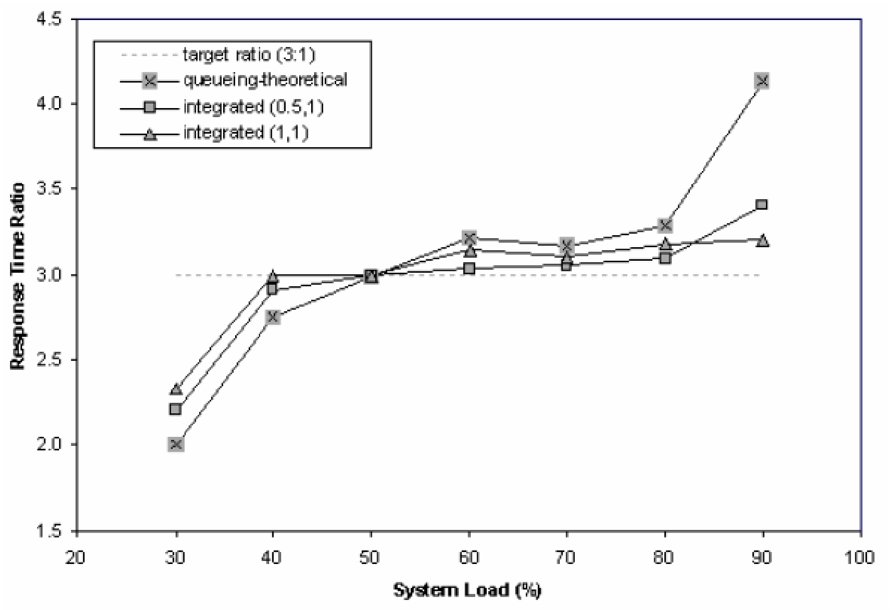
classes is kept to be 3:1. As shown by Figure 6.4(b), the expected response time ratio can be achieved when the system load is between 30% and 80%. Thus, the proportional response time differentiation is achieved.

c) Integrated Process Allocation

The objective of the integrated approach is to reduce the difference between the target response time ratio and the achieved response time ratio of two classes due to the queueing-theoretical approach. Meanwhile, it also aims to reduce the variance of the ratios due to the queueing-theoretical approach.

Figure 6.5(a) depicts the achieved response time ratio due to the integrated approach and the queueing-theoretical approach, respectively.. The arrival rate ratio of two classes is 3:1 and the differentiation weight ratio is set to be 1:2. In the integrated approach, two different sets of control parameter were adopted. As we observe from the figure, the performance of PID controller is quite sensitive to parameter settings. Actually, it is a non-trivial task to tune the three parameters to get good performance of proportional differentiation. Like others in [BKKL03], we assign the same value to the three PID parameters. Integrated (α, β) gives the results due to the feedback parameter settings: the PID parameters are set as KP1 = KI1 = KD1 = α for class 1 and KP2 = KI2 = KD2 =β for class 2. Note that a set of good parameters for one class may not be effective for the other, and vice versa.

From Figure 6.5(a), we can observe that the integrated approach with both feedback parameter settings significantly outperforms the queueing-theoretical approach with respect to the response time differentiation proportionality. When the arrival rate is below



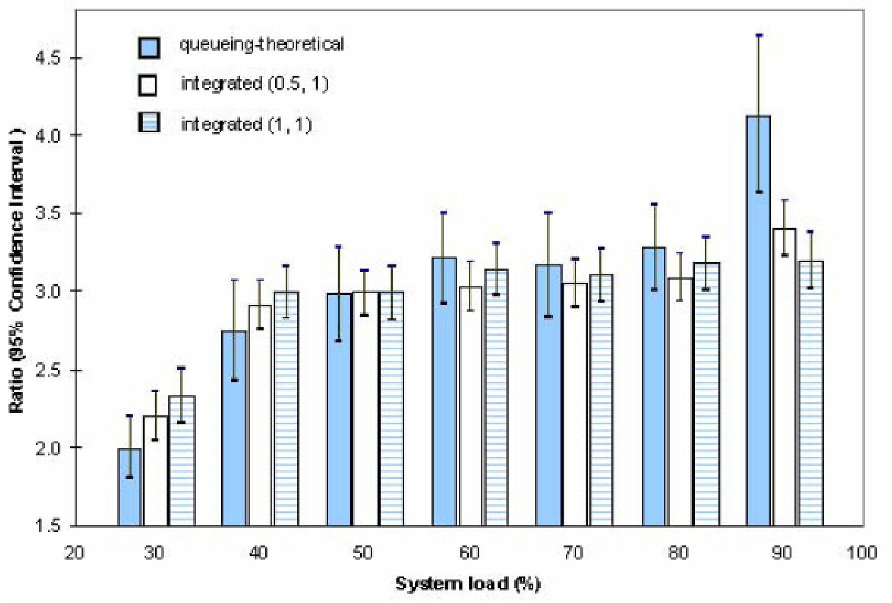


Figure 6.6(a-b): Achieved average response time ratio and 95% confidence intervals for integrated process allocation (δ_1: δ_2=1:3)

30%, the expected response time proportionality cannot be achieved. This is explained by the fact that when the workload is light, there is almost no queueing delay observed in all traffic queues. Because the scheduling is work conserving and non-preemptive, service differentiation is not feasible under certain light load conditions [CDDS02]. In reality, differentiation may not be necessary during light load conditions since the resources are sufficient. Therefore, in the following diagrams, we will not present the results when the workload is less than 30%. When the system load is close to system capacity, say at 90%, the queueing-theoretical approach generates very poor proportionality. This can be explained by the fact that as the system load is close to its capacity, the impact of the variance of interarrival times on queueing delay dominates and thus queueing delay in all traffic queues increase significantly. This affects the controllability of the queueing-theoretical process allocation approach. On the other hand, the integrated approach with feedback control is able to maintain desirable differentiation proportionality.

Figure 6.5(b) depicts the achieved 95% confidence intervals due to the approaches, respectively. It shows that the integrated approach not only improves the differentiation proportionality robustness in terms of the achieved mean response time ratios, but also significantly outperforms the queueing-theoretical approach with much shorter confidence intervals.

Figure 6.6(a) depicts the achieved response time ratio due to the two approaches, respectively. The arrival rate ratio of two classes is 3:1 and the differentiation weight ratio is now changed to 1:3. It can be observed that the integrated approach outperforms the queueing-theoretical approach with respect to the achieved response time ratios. In particular, the integrated approach can maintain desirable differentiation proportionality during heavy load conditions. Figure 6.6(b) further depicts the achieved 95% confidence intervals due to the approaches. It is obvious that the integrated approach generates much short confidence intervals than the queueing-theoretical approach.

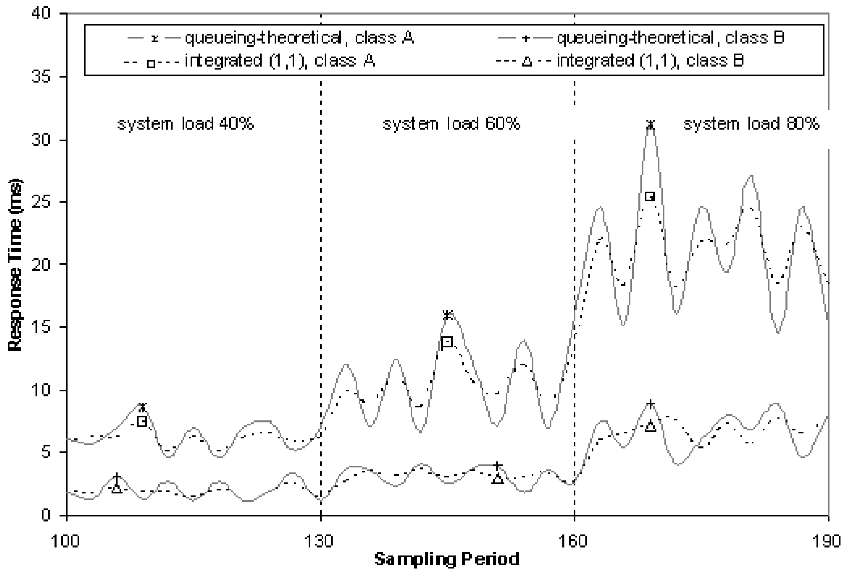


Figure 6.7: A microscopic view of response time

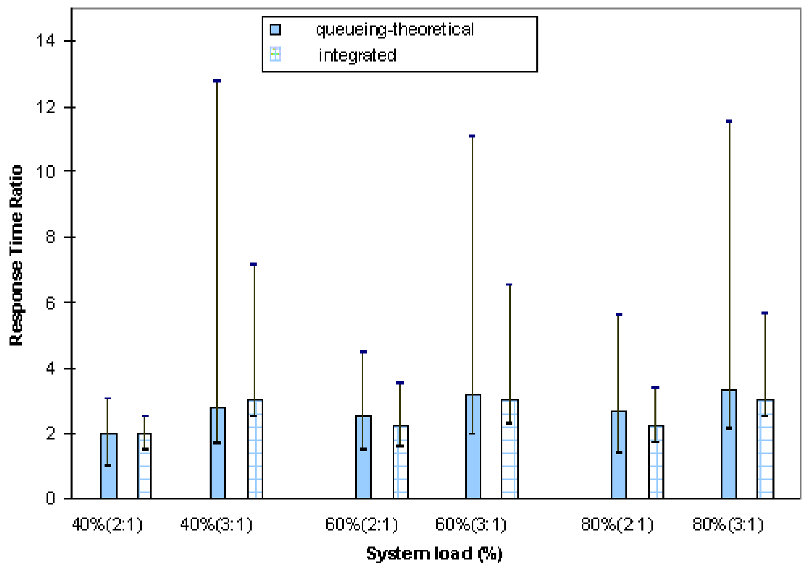


Figure 6.8: The variance of response time ratio

Figure 6.7 shows a microscopic view of the response time of individual requests of the two classes due to the two approaches, when the system workload is 40%, 60%, and 80%, respectively. The target response time ratio of class A to class B is 3:1. The experiments were run for 100 sampling periods for warming up and then the data was collected for 30 sampling periods at each of three workload conditions. Obviously, we can observe that the integrated approach achieves more consistent results during different sampling periods at various workload conditions.

Figure 6.8 further quantitatively depicts the variance of the proportionality due to the two approaches. At each of the three workload conditions (40%, 60%, 80%), we conducted experiments by using a two-class workload with the target response time ratio 2:1 and 3:1, respectively. The upper line is the 95th percentile; the bar is the mean; and the lower line is the 5th percentile. We can observe that the integrated approach can significantly reduce the variance. For example, when the workload is 80% and the target proportionality is 2, the difference between the 95th and the 5th percentile is 1.7 and 4.2 due to the integrated approach and the queueing-theoretical approach, respectively. Furthermore, the mean is 2.2 and 2.7, respectively. At 80% workload condition, when the target ratio is 3, the difference between 5th and 95th is 3.1 and 9.3, and the mean is 3 and 3.3, due to the integrated approach and the queueing-theoretical approach, respectively.

We conducted a wide ranger of sensitivity analyses. We varied the number of classes, the arrival rate ratio of the classes, and the differentiation weight ratio of the classes. While we do not have space to present all of the results, it worths note that we did not reach any significantly different conclusion regarding to the differentiation proportionality achieved by the integrated approach.

Conclusion

Providing proportional response time to different client classes is an important and challenging issue. It is important because proportional model is a popular relative DiffServ model and response time is a fundamental QoS metric on Web servers. It is challenging because the conventional application-level process allocation approaches lack fine-grained control of resource allocation and are insensitive to the bursty Internet traffic.

CHAPTER VII

CONCLUSION AND FUTURE WORK

Conclusion

In this dissertation, we study the proxy server based multipath connection (PSMC). First, a proxy server based overlay network using a set of intermediate connection relay proxy servers is designed and implemented. This overlay network is used in a Secure Collective Defense system (SCOLD) to defend against DDoS attacks. The BIND9 DNS server and its DNS update utilities are enhanced to support new DNS entries with indirect routing information. The indirect route is implemented by utilizing IP Tunnel. Protocol software for supporting the establishment of indirect routes based on the new DNS entries is developed for Linux systems. The experimental results validate the capability of PSMC in enhancing network security and reliability.

Second, a proxy server based multipath protocol is designed and implemented by enhancing the existing TCP/IP protocol to effectively distribute, transport and reassemble network packets over the multiple indirect paths between two end hosts. We modify the Linux kernel to support the enhanced TCP/IP protocols. On the sender side, the IP layer is enhanced to stripe packets across multiple paths. The TCP congestion window control is also revised for higher throughput. On the receiver side, the TCP layer is enhanced with a double buffer to solve the TCP packet persistent reordering problem over multiple paths. A communication channel is set up between sender and receiver for exchanging network traffic information. The PSMC support both TCP and UDP, which enable PSMC to support multimedia applications in today's Internet. The experimental results show that PSMC can effectively utilize the aggregate bandwidth from multiple routes and significantly improve the network performance.

Third, proxy server selection algorithms are developed to select a subset of proxy servers from a large set of available proxy servers to meet various object functions and constraints. When there are hundreds of proxy servers available, the disjoint paths are more desirable because the route correlation can be reduced and network reliability and throughput can be improved. The experimental results show that different sever selection may result in significantly different network performance, and the heuristic (genetic) algorithms we proposed can yield satisfactory results for the NP-Complete problems.

Forth, resource allocation schemes on the end server and server cluster are designed and implemented to provide proportional differentiated services. These schemes are based on the queueing theory and feedback control theory. A process allocation approach on Apache Web server is presented for proportional responsiveness differentiation. A two-tier resource allocation approach for proportional slowdown differentiation on cluster-based network is also presented.

Combining the multipath on network with service differentiation on the end server, a comprehensive solution for various QoS and security related applications can be provided.

The research result and insight gained from PSMC could have broader impact on the protocols and security of today's Internet.

Future Work

For the proxy server based overlay network (SCOLD), there are a set of challenging problems.

1) How should the Internet community form trust relationships and coordinate with each other?

2) How to detect and deal with the compromised proxy server nodes.

3) How to effectively manage and maintain a trusted proxy server list on clients?

4) How to improve the security on SCOLD itself and prevent future attacks and misuse?

For the proxy server based multipath connection (PSMC), the future works are listed below.

1) With more network resources available, how to control the aggressive or malicious users?

2) How to fairly distribute the network resources among the Internet participants?

3) Study more efficient double buffer management scheme.

4) Derive a TCP throughput formula for multipath connection.

5) Derive a TCP latency theoretical result for multipath connection with double buffer.

6) How to design better proxy server / alternate path selection algorithm

7) Client clustering algorithms with multiple proxy servers

8) Combine multipath and QoS to get more results

For the proportional differentiation provisioning on end server, the future works are listed below.

1) Design and utilize better resources management units, like resource container.

2) Design better processing rate allocation schemes on Web server, FTP, and cluster network.

BIBLIOGRAPH

[AAJP04] Aditya A., Jeffrey P., et. al., “A Comparison of Overlay Routing and Multihoming Route Control”, In Proceedings of the ACM SIGCOMM 2004.

[ABan96] A. Banerjea, “Simulation study of the capacity effects of dispersity routing for fault tolerant realtime channels”, Proceedings of ACM SIGCOMM, 1996.

[AKA] Akamai.com, http://www.akamai.com

[ANSD99] A. Nasipuri and S.R. Das, “On-Demand Multipath Routing for Mobile Ad Hoc Networks”, Proceedings of IEEE ICCCN'99, Boston, MA, Oct. 1999, pp. 64-70.

[BBN] BBN Technologies, "Applications that participate in their own defense," http://www.bbn.com/infosec/apod.html

[BIND9] DNS BIND 9, http://www.isc.org/products/BIND/

[BLMG99] B. Li, M. J. Golin, G. F. Ialiano, and X. Deng, "On the optimal placement of web proxies in the internet," Proc. of IEEE INFOCOM, Mar. 1999.

[BKJW00] B. Krishnamurthy, and J. Wang, "On network-aware clustering of web clients," Proc. of ACM SIGCOMM, Aug. 2000.

[BKKL03] B. Ko, K. Lee, K. Amiri, and S. Calo. Scalable service differentiation in a shared storage cache. In Proc. 23rd IEEE Int’l Conf. on Distributed Computing Systems (ICDCS), 2003.

[BYPM02] B. Yang and P. Mohapatra. Multicasting in differentiated service domains. In Proc. of IEEE Globecom, 2002.

[CCac02] Christian Cachin, et al. “Secure Intrusion-tolerant Replication on the Internet”, Proc. Intl. Conference on Dependable Systems and Networks, 2002.

[CCas02] C. Casetti, et. al., "TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks", In Wireless Networks Journal 8, 467-479, 2002

[CDDS99] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. In Proc. ACM SIGCOMM, 1999.

[CDDS02] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. IEEE/ACM Trans. on Networking, 10(1):12–26, 2002.

[Chow00] E. Chow, “network measurement”, technical report in UCCS 2000

[Chow02] A. Cearns, “A2D2”, master thesis in UCCS 2003

[Chow04] Edward Chow, Yu Cai, David Wilkinson, and Ganesh Godavari, “Secure Collective Defense System”, In Proceedings of GlobeCom 2004.

[CLSM90] C-L Li, S.T. McCormick, D. Simchi-Levi, “The complexity of finding two disjoint paths with min-max objective function”, Discrete Applied Mathematics, Vol. 26, No. 1, pp. 105-115, January 1990.

[CPER99] C.E. Perkins and E.M. Royer. "Ad hoc On-Demand Distance Vector Routing." In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and applications, February 1999, pp. 90-100.

[CPKS82] C.H. Papadimitriou and K. Steiglitz, “Combinatorial Optimization Algorithms and Complexity”, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[CPro] cprobe, http://cs-people.bu.edu/carter/tools/Tools.html

[CSim] Christoph Simon, “How to to use more than one independent Internet connection”, http://www.ssi.bg/~ja/nano.txt

[DDU] Dynamic DNS update, RFC 2136, http://www.faqs.org/rfcs/rfc2136.html

[DETO] Detour, http://www.cs.washington.edu/research/networking/detour/

[DJDM96] D.B. Johnson and D.A. Maltz, “Dynamic Source Routing in Ad HocWireless Networks”, In Mobile Computing, 1996, pp. 153-181.

[DMTJ] D. Mosberger and T. Jin. Httperf: a tool for measuring Web server performance. http://www.hpl.hp.com/personal/David Mosberger/httperf.html.

[DSAP01] D. X. Song and A. Perrig, "Advanced and authenticated marking schemes for IP Traceback," IEEE Infocom 2001.

[DSEC] DNSSEC, http://www.dnssec.net/

[DSRN91] D. Sidhu, R. Nair, and S. Abdallah, “Finding Disjoint Paths in Networks”, Proceedings of ACM SIGCOMM'91, Zurich, Switzerland, Sep. 1991, pp. 43-51.

[DWil04] David Wilkinson, Edward Chow and Yu Cai, “Enhanced Secure Dynamic DNS Update with Indirect Route”, In Proceedings of the IEEE Workshop on Information Assurance, 2004.

[DYNA] Information Sciences Institute, “Dynabone”, http://www.isi.edu/dynabone

[EDDI] Eddie, Enhanced DNS Server, http://eddie.sourceforge.net/lbdns.html

[EONY95] E. Oki and N. Yamanaka, “A recursive matrix calculation method for disjoint path search with hop link number constraints”, IEICE Trans. Commun., Vol. E78-B, No.5, pp. 769-774, May 1995.

[ERun] Eddie Runner, “Multi Path Interference”, http://www.mmxpress.com/technical/multipath.htm

[EYYM] E. Yilmaz and Y. Manzano, "Surveying Formal and Practical Approaches for Optimal Placement of Replicas on the Web", http://websrv.cs.fsu.edu/research/reports/TR-020701.pdf

[FGlo92] F. Glover, et.al., “Network Models in Optimization and Their Applications in Practice”, Wiley-Interscience, 1992

[Fran05] Frank Waltson, “Multipath”, master thesis in UCCS 2005.

[GFJP02] G. F. Franklin, J. D. Powell, and A. Emami-naeini. Feedback control of dynamics systems. Prentice Hall, 4th edition, 2002.

[GBPD99] G. Banga, P. Druschel, and J. Mogul. Resource containers: A new facility for resource management in server systems. In Proc. USENIX Symposium on Operating System Design and Implementation, 1999.

[GHer]Glenn Herrin,” Linux IP Networking”, http://www.cs.unh.edu/cnrg/gherrin/

[GITM] GT-ITM, http://www.cc.gatech.edu/projects/gtitm/

[GYPM03] Guo, Y., F. A. Kuipers and P. Van Mieghem, "Link-Disjoint Paths for Reliable QoS Routing", International Journal of Communication Systems, vol. 16, pp. 779-798, 2003.

[GYFK03] Guo, Y., F. A. Kuipers and P. Van Mieghem, "Link-Disjoint Paths for Reliable QoS Routing", International Journal of Communication Systems, vol. 16, pp. 779-798, 2003.

[HAdi96] H. Adiseshu, et. al., “A reliable and scalable striping protocol”, In Proceedings of ACM SIGCOMM, 1996.

[HHsi02] H. Hsieh, et. al., “ptcp: An end-to-end transport layer protocol for striped connections”, In Proceedings of IEEE ICNP, 2002.

[HSiv00] H. Sivakumar, et. al., “PSockets: The case for application-level network striping for data intensive applications using high speed wide area networks”. In Supercomputing, 2000.

[HZHT01] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation for cluster-based network servers. In Proc. IEEE INFOCOM, pages 679–688, 2001.

[HWel] Harald Welte, “The journey of a packet”, http://gnumonks.org/ftp/pub/doc/packet-journey-2.4.html

[IChi] Internet in China, http://austlii.edu.au/~graham/hkitlaw/Choy_and_Cullen.html

[ICRR99] I. Cidon, R. Rom, and Y. Shavitt, “Analysis of Multi-Path Routing”, IEEE/ACM Transactions on Networking, vol. 7, no. 6, Dec. 1999, pp. 885-896.

[IPIP] “IPIP tunnel”, http://www.europe.redhat.com/documentation/HOWTO/Net-HOWTO/x1284.php3

[IPSE] IPSec, http://www.ietf.org/html.charters/ipsec-charter.html

[IPV6] GUARDINI, I., et. al. “IPv6 Operational Experience within the 6bone”. In Proc. Internet Society Conference 2000

[JAna] Julian Anastasov, “linux kernel patches”, http://www.ssi.bg/~ja/

[JAMD98] J. Almeida, M. Dabu, A. Manikutty, and P. Cao. Providing differentiated levels of services in Web content hosting. In Proc. ACM SIGMETRICS Workshop on Internet Server Performance, pages 91–102, 1998.

[JChe98] Johnny Chen, “New Approaches to Routing for Large-Scale Data Network”, PhD Thesis, Rice University, 1998.

[JMir03] Jelena Mirkovic, et al. “A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms”, UCLA Technical Report, 2003

[JKoz92] John R. Koza, “Genetic Programming”, MIT Press, 1992.

[JPVF98]J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: a simple model and its empirical validation. ACMSIGCOMM, September 1998

[JSRT84] J.W. Suurballe and R.E. Tarjan, “A Quick Method for Finding Shortest Pairs of Disjoint Paths”, Networks, Vol. 14, pp. 325-333, 1984.

[JSuu74] J.W. Suurballe, “Disjoint Paths in a Network”, Networks, Vol. 4, pp. 125-145, 1974.

[JY00] J. Yan et al., "The XenoService – A distributed defeat for DDoS", In Proceedings of ISW 2000.

[JWCX04] J. Wei, C.-Z. Xu, and X. Zhou. A robust packet scheduling algorithm for proportional delay differentiation services. In Proc. of IEEE Globecom, 2004.

[KLai01] K. Lai et.al., "Nettimer: A Tool for Measuring Bottleneck Link Bandwidth", In Proceedings of the USENIX 2001

[KSHT02] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated resource management for cluster-based Internet services. In Proc. of USENIX OSDI, pages 225–238, December 2002.

[LAgg] IEEE 802, “IEEE 802.3ad Link Aggregation”, http://grouper.ieee.org/groups/802/3/ad/index.html

[LEJH99] L. Eggert and J. Heidemann. Application-level differentiated services for Web servers. World Wide Web Journal, 3(2):133–142, 1999.

[LKle76] L. Kleinrock. Queueing Systems, Volume II. John Wiley and Sons, 1976.

[LQVP01] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of web server replicas," Proc. of IEEE INFOCOM, Mar. 2001

[LZZZ02] Lianfang Zhang, Zenghua Zhao, Yantai Shu, Lei Wang, and Oliver W.W. Yang; ``Load Balancing of Multipath Source Routing in Ad Hoc Networks''; In Proceedings of IEEE International Conference on Communications (ICC 2002), 2002.

[MADK01] M. Arlitt, D. Krishnamurthy, and J. Rolia. Characterizing the scalability of a large Web-based shopping system. ACM Trans. on Internet Technology, 1(1):44–69, 2001.

[MAPD00] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: a mechanism for resource management in cluster-based network servers. In Proc. ACM SIGMETRICS, pages 90–101, 2000.

[MAZU] Mazu Networks, "Dynamically Provisioned Monitoring, traffic master" http://www.mazunetworks.com/white_papers/provmon-toc.html

[MBON] ERIKSSON, H. “Mbone: The Multicast Backbone”. Communications of the ACM 37, 8 (1994), 54–60.

[MEAB03] M. El-Gendy, A. Bose, and K. G. Shin. Evolution of the Internet QoS and support of soft real-time applications. In Proc. of the IEEE, July 2003.

[MGDJ79] M. Gary, D. Johnson, “Computers and intractability, a guide to the theory of NP-completeness”, W.H. Freeman Press 1979

[MLJL01] M. K. H. Leung, J. C. S. Lui, and D. K. Y. Yau. Adaptive proportional delay differentiated services: Characteriza-tion and performance evaluation. IEEE/ACM Trans. on Networking, 9(6):908–817, 2001.

[MMat97] M. Mathis, et. al., “The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm.” Computer Communication Review, volume 27, number3, 1997.

[MRic03] M. Richardson, “A method for storing IPsec keying material in DNS”, http://www.sandelman.ottawa.on.ca/SSW/ietf/ipsec/key/draft-richardson-ipsec-rr.html, 2003

[MTMS04] M. M. Teixeira, M. J. Santana, and R. H. C. Santana. Using adaptive priority scheduling for service differentiation QoS-aware Web servers. In Proc. IEEE 23rd Int’l Conf. on Performance, Computing, and Communications (IPCCC), pages 279–285, 2004.

[Mcast]Multicast, http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ipmulti.htm

[MZha04] M. Zhang, et. al., “A Transport Layer Approach for Improving End-to-End Performance and Robustness Using Redundant Paths”, In Proc. of the USENIX 2004 Annual Technical Conference. 2004.

[NBRF99] N. Bhatti and R. Friedrich. Web server support for tiered services. IEEE Network, 13(5):64–71, 1999.

[NMax75] N. F. Maxemchuk, “Dispersity Routing in Store and Forward Networks”, Ph.D. thesis, University of Pennsylvania, 1975.

[NEWS-1] Internetnews, “Massive DDoS Attack Hit DNS Root Servers”, http://www.internetnews.com/

[NEWS-2] Computer World, http://www.computerworld.com/printthis/2004/0,4814,93977,00.html

[NMax75] N. F. Maxemchuk, “Dispersity Routing in Store and Forward Networks”, Ph.D. thesis, University of Pennsylvania, 1975.

[NS2] NS2, http://www-mash.cs.berkeley.edu/ns

[NSUP] Nsupdate, http://www.linuxforum.com/man/nsupdate.8.php

[NTBB99] N. Taft-Plotkin, B. Bellur, and R. Ogier, “Quality-of-Service Routing Using

Maximally Disjoint Paths", Proceedings of IEEE IWQoS'99, London, UK, Jun. 1999, pp. 119-128.

[OSI] OSI, "Open System Interconnection Protocols", http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/osi_prot.htm

[OSSL] OpenSSL, http://www.openssl.org

[Path] pathchar, http://www.caida.org/tools/utilities/others/pathchar/

[PKDR00] P. Krishnan, D. Raz, and Y. Shavitt, "The cache location problem," ACM/IEEE Transactions on Networking, vol. 8, no. 5, Oct. 2000.

[PRAK00] Pablo Rodriguez, Andreas Kirpal, Ernst W. Biersack, "Parallel-Access for Mirror Sites in the Internet", Proceeding of Infocom, 2000

[PSSH02] P. J. Shenoy, S. Hasan, P. Kulkarni, and K. Ramamritham. Middleware versus native OS support: architectual considerations for supporting multimedia applications. In Proc. IEEE Real-Time Technology and Application Symposium, 2002.

[PSar02] Pasi Sarolahti, Linux TCP, http://www.cs.helsinki.fi/u/kraatika/Courses/sem02a/Linux-TCP.pdf, 2002

[RATM93] R.K. Ahuja, T.L. Magnanti, "Network flows", Prentice-Hall, 1993

[RON01] D. G. Andersen, et al., "Resilient Overlay Networks," In Proceedings of 18th ACM SOSP, October 2001.

[RBha94] R. Bhandari, "Optimal Diverse Routing in Telecommunication Fiber Networks", Proc. IEEE INFOCOM 1994, Toronto, Ontario, Canada, Vol.3, pp.1498-1508, June 1994

[RONS89] R. Ogier and N. Shacham, "A distributed algorithm for finding shortest pairs of disjoint paths," in Proceedings of IEEE INFOCOM 1989

[ROVR93] R. Ogier, V. Rutenburg, and N. Shacham, “Distributed Algorithms for Computing Shortest Pairs of Disjoint Paths”, IEEE Transactions on Information Theory, vol. 39, no. 2, Mar. 1993, pp. 443-455.

[RSTP] RSTP, http://www.cs.columbia.edu/~hgs/rtsp/

[RSVP] RSVP, http://www.isi.edu/div7/rsvp/rsvp.html

[SBDB98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. IETF RFC 2475, 1998.

[SBoh04] S. Bohacek, et. al. “A New TCP for Persistent Packet Reordering”, In Transactions on Networking, 2004.

[SCCE00] S. Chandra, C. S. Ellis, and A. Vahdat. Differentiated multimedia Web services using quality aware transcoding. In Proc. IEEE INFOCOM, pages 961–968, 2000.

[SDU] Secure DNS update, http://www.faqs.org/rfcs/rfc3007.html

[SJCJ00] S. Jamin, C. Jin, D. Raz, Y. Shavitt, and L. Zhang, "On the placement of internet instrumentation," Proc. of IEEE INFOCOM, Mar. 2000

[SLCW99] S.W. Lee and C. S. Wu, .”A k-best paths algorithm for highly reliable communication networks”., IEICE Trans. On Commun., Vol. E82-B, No.4, pp.586-580, April 1999.

[SLee00] S. Lee et. al., “Split Multipath Routing with Maximally Disjoint paths in Ad Hoc Networks”, Technical Report, University of California, 2000.

[SLMG00-1] S. Lee and M. Gerla. “Split Multipath Routing with Maximally Disjoint paths in Ad Hoc Networks”. In Technical Report in University of California, 2000.

[SMJG96] S. Murthy and J.J. Garcia-Luna-Aceves, “Congestion-Oriented Shortest

Multipath Routing", Proceedings of IEEE INFOCOM'96, San Francisco, CA, Mar. 1996, pp. 1028-1036.

[SLow02] Steven Low, TCP Congestion Control, CalTech Tech Report, 2002.

[SLJL04] S. C. M. Lee, J. C. S. Lui, and D. K. Y. Yau. A proportional-delay diffserv-enabled Web server: admission control and dynamic adaptation. IEEE Trans. on Parallel and Distributed Systems, 15(5):385–400, 2004.

[SNA79] J. P. Gray and T. B. McNeill. SNA multiple-system networking. IBM Systems Journal, 18(2):263–297, 1979.

[SOCKS] SOCKS proxy server, http://www.tldp.org/HOWTO/Firewall -HOWTO-11.html

[SSav99] Stefan Savage, et al. "Detour: a Case for Informed Internet Routing and Transport," IEEE Micro, pp. 50-59, v 19, no 1, 1999.

[SSav00] Stefan Savage, et al. "Practical network support for IP Traceback," In Proceedings of 2000 ACM SIGCOMM Conference, Aug. 2000.

[SSin]Shweta Sinha, TCP tutorial, http://www.ssfnet.org/Exchange/tcp/tcpTutorialNotes.html

[STA4] StacheldrahtV4, http://cs.uccs.edu/~scold/ddos

[SVJG01] S. Vutukury and J.J. Garcia-Luna-Aceves, "MDVA: A distance-vector multipath routing protocol," Proceedings of the IEEE INFOCOM, pp. 557–564, 2001.

[SVut01] S. Vutukury et.al., "MDVA: A distance-vector multipath routing protocol," Proceedings of the IEEE INFOCOM, 2001.

[TAKS02] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for Web server end-systems: a control-theoretical approach. IEEE Trans. on Parallel and Distributed Systems, 13(1):80–96, 2002.

[TGMP01] T. M. Gil and M. Poleto, "MULTOPS: a data-structure for bandwidth attack detection," In Proceedings of 10th Usenix Security Symposium, August 2001.

[THac02] T. Hacker, et. al., “The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network”, In Proceedings of IPDPS, 2002.

[Trac] traceroute, http://www.traceroute.org/

[TNgu03] T. Nguyen et.al., “Path diversity with forward error correction system for packet switched networks”, In Proceedings of IEEE INFOCOM, 2003.

[VCEC01] V. Cardellini, E. Casalicchio, M. Colajanni, and M. Mambelli. Web switch support for differentiated services. ACM SIGMETRICS Performance Evaluation Review, 29(2):14–19, 2001.

[VPax] Vern Paxson, “Measurements and Analysis of End-to-End Internet Dynamics ”, Ph.D. dissertation at UC Berkley.

[VOIP] VOIP, http://www.fcc.gov/voip/

[VPN] Virtual Private Network, http://www.vpnc.org/

[WA99] W. Adjie, et al., “The design and implementation of an intentional naming system”, Operating Systems Review, vol.35, pp. 186-201, 1999.

[WZJG98] W.T. Zaumen and J.J. Garcia-Luna-Aceves, “Loop-Free Multipath Routing Using Generalized Diffusing Computations”, Proceedings of IEEE INFOCOM' 98, San Francisco, CA, Mar. 1998, pp. 1408-1417.

[XBON] TOUCH, J., AND HOTZ, S. “The X-Bone”. In Proc. 3rd Global Internet Mini-Conference

[XCPM02] X. Chen and P. Mohapatra. Performance evaluation of service differentiating Internet servers. IEEE Trans. on Computers, 51(11):1,368–1,375, 2002.

[XZCX04] X. Zhou and C.-Z. Xu. Harmonic proportional bandwidth allocation and scheduling for service differentiation on streaming servers. IEEE Trans. on Parallel and Distributed Systems, 15(9):838–551, 2004.

[XZJW04] X. Zhou, J. Wei, and C.-Z. Xu. Modeling and analysis of 2D service differentiation on e-Commerce servers. In Proc. of IEEE 24th Int’l Conf. on Distributed Computing Systems (ICDCS), pages 740–747, March 2004.

[XZYC04] X. Zhou, Y. Cai, G. K. Godavari, and C. E. Chow. An adaptive process allocation strategy for proportional responsiveness differentiation on Web servers. In Proc. IEEE 2nd Int’l Conf. on Web Services (ICWS), July 2004.

[YCai05] Yu Cai, Edward Chow. “Proxy Server Based Multipath Connection”, Technical Report, http://cs.uccs.edu/scold/psmc.pdf, 2005

[YHRG04] Y. Huang and R. Gu. A simple fifo-based scheme for differentiated loss guarantees. In Proc. IWQoS, 2004.

[YZha02] Y. Zhang, et. al., “On the characteristics and origins of Internet flow rates”. In Proceedings of SIGCOMM, 2002.

[ZEBE] Zebedee, http://www.winton.org.uk/zebedee/

[ZWJC96] Z. Wang and J. Crowcroft, “QoS Routing for supporting Multimedia Applications”, IEEE J. Selected Areas in Communications, Vol. 14, No.7, pp. 1228-1234, September 1996.

APPENDIX A

SCOLD USER MANUAL

The SCOLD network is easily configured to illustrate an application layer overlay network and to guide against various types of DDoS attacks. This user manual explains the steps required to configure and start the SCOLD network. Sample demonstration scripts and files are also provided as a step-by-step guide on how SCOLD works. The set up of the SCOLD test-bed includes three main components:

- The sender network
- The receiver network
- The proxy servers

A1 Overview

We have set up several different SCOLD testbeds for different research purposes. Figure A1 is one of the SCOLD testbeds. All machines in the testbed are VMWare virtual machines based. The virtual machine files are under d:/vmware/ycai directory on ardor.uccs.edu machine. To start a VMWare virtual machine, just open the virtual machine file in the VMWare GUI interface. For more information on VMWare, please refer to http://www.vmware.com/support/pubs/ws_pubs.html.

Figure A2 is another SCOLD testbeds. All machines are VMWare virtual machines and resides under d:/vmware/ycai2 directory on ardor.uccs.edu machine. Figure A3 is the third SCOLD testbed on real machines. The testbed is shared with IDIP project.

A2 Indirect Routing using IP Tunnel

1) Install IP tunnel package.

By default, Redhat Linux (version 9 or later) has IP tunnel package installed. You can check it by running:

```
"ip tunnel add tunl1 mode ipip remote 192.168.1.1"
```

If you can successfully add a new IP tunnel (see Section A2 - 3 “Verify the configuration and IP tunnel”), then you are OK. Otherwise, you need to recompile Linux kernel to enable IP tunnel. The following Linux kernel compile options show that IP tunneling is to be built into the kernel as opposed to be compiled as a dynamic kernel module.

Linux Kernel Compile Options:

```
Networking options  --->
   [*] TCP/IP networking
   [*] IP: forwarding/gateway
   ....
   <*> IP: tunneling
```

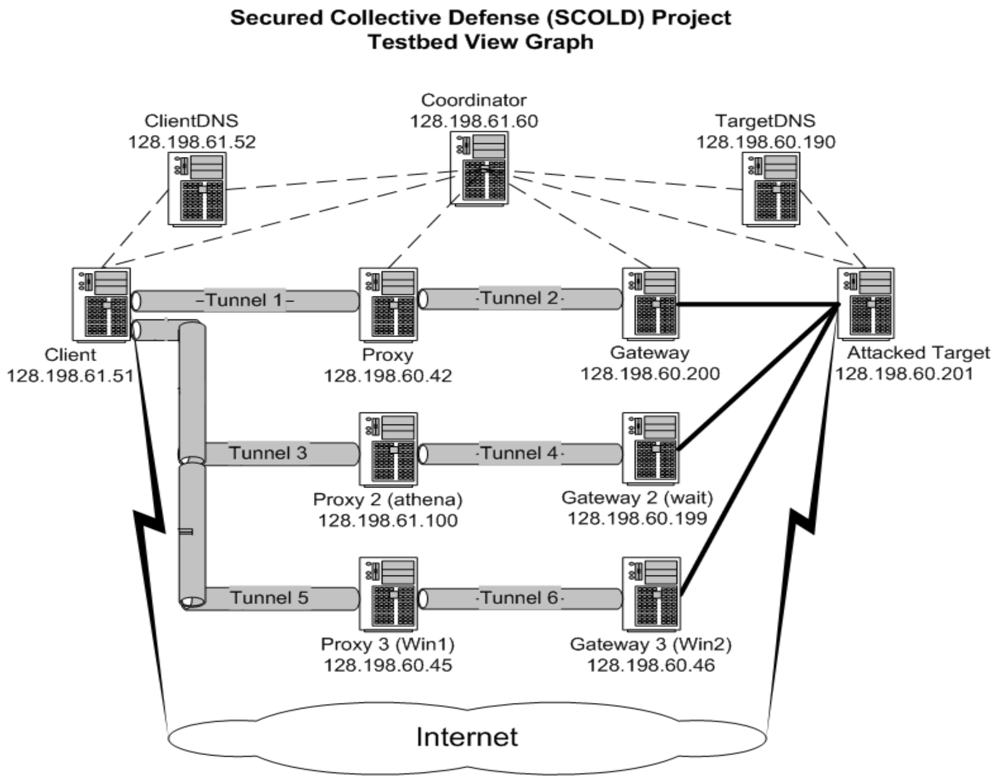


Figure A1: SCOLD testbed 1

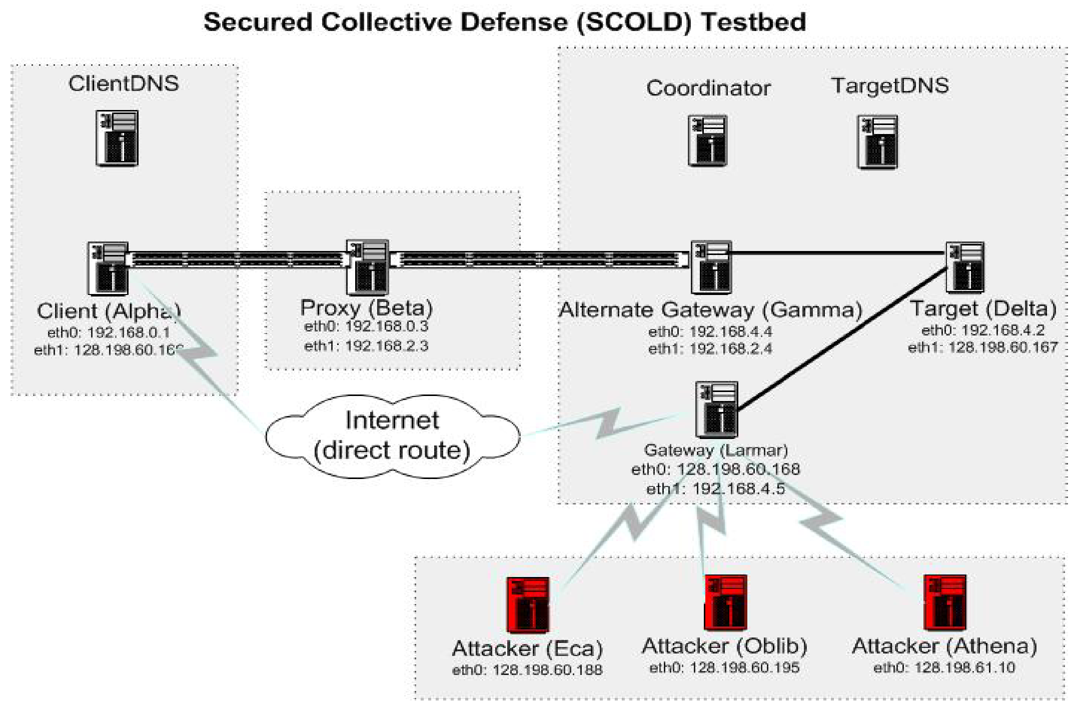


Figure A2: SCOLD testbed 2

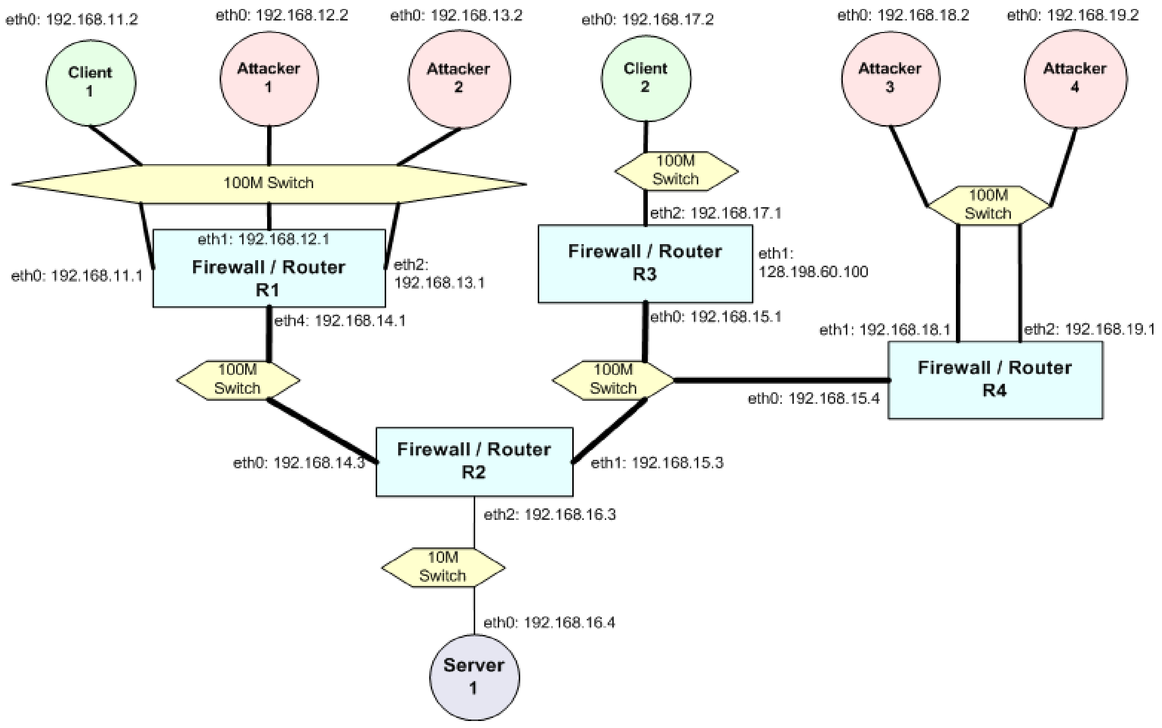


Figure A3: SCOLD testbed 3

2) Set up IP tunnel

Below are scripts on each type of machine in SCOLD to set up IP tunnel in testbed 1.

a) Script on the client to setup tunnel. The IP address section in the script need to be customized as needed. For more information on IP command, please refer to http://linux-ip.net/gl/ip-cref/. Note in the script, the client has no information about the gateway IP. Client only knows the target server IP and designated proxy server IP.

```
#!/bin/sh

#define var
client_ip=128.198.61.51
client_gw=128.198.61.1
proxy_ip=128.198.60.42
target_ip=128.198.60.201
tunl=tunl1

#config tunnel between client and proxy
ip tunnel add $tunl mode ipip remote $proxy_ip dev eth0
ifconfig $tunl $client_ip
ip link set $tunl up
ip route add $proxy_ip via $client_gw dev $tunl onlink

#route traffic between client and target through tunnel
ip route add $target_ip via $client_gw dev $tunl onlink
```

b) Script on the proxy servers. Note that the proxy servers have information about the client IP, gateway IP and target IP.

```
#!/bin/sh

iptables -F
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
#enable ip forwarding
echo "1" > /proc/sys/net/ipv4/ip_forward

#define var
client_ip=128.198.61.51
proxy_ip=128.198.60.42
proxy_gw=128.198.60.1
gw_ip=128.198.60.200
target_ip=128.198.60.201

#config tunnel between proxy and client
tunl=tunl1
ip tunnel add $tunl mode ipip remote $client_ip dev eth0
```

```
ifconfig $tunl $proxy_ip
ip link set $tunl up
ip route add $client_ip via $proxy_gw dev $tunl onlink

#config tunnel between proxy and gateway
tunl=tunl2
ip tunnel add $tunl mode ipip remote $gw_ip dev eth0
ifconfig $tunl $proxy_ip
ip link set $tunl up
ip route add $gw_ip via $proxy_gw dev $tunl onlink

#route between proxy and target through tunnel
ip route add $target_ip via $proxy_gw dev $tunl onlink
```

c) Script on the gateway in target server network. Note that the client IP, target IP and proxy IP are needed in the script.

```
#!/bin/sh

iptables -F
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
echo "1">/proc/sys/net/ipv4/ip_forward

#define var
client_ip=128.198.61.51
proxy_ip=128.198.60.42
gw_ip=128.198.60.200
gw_gw=128.198.60.129
tunl=tunl2

#config tunnel between gateway and proxy
ip tunnel add $tunl mode ipip remote $proxy_ip dev eth0
ifconfig $tunl $gw_ip
ip link set $tunl up
ip route add $proxy_ip via $gw_gw dev $tunl onlink

#route traffic between client and gateway through tunnel
ip route add $client_ip via $gw_gw dev $tunl onlink
```

d) Script on the target server.

```
#!/bin/sh

#define var
client_ip=128.198.61.51
gw_ip=128.198.60.200
```

```
#route between client and target through gateway
ip route add $client_ip via $gw_ip dev eth0 onlink
```

3) Verify the configuration and IP tunneling

After configuring machines in the testbed with the above scripts, you should be able to see the ip tunnel devices using the following commands.

```
[root@client root]# ifconfig
eth0 Link encap:Ethernet HWaddr 00:0C:29:47:59:30
inet addr:128.198.61.51 Bcast:128.198.61.63 Mask:255.255.255.192
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:116492 errors:0 dropped:0 overruns:0 frame:0
TX packets:543 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:100
RX bytes:8552726 (8.1 Mb) TX bytes:48894 (47.7 Kb)
Interrupt:18 Base address:0x10a0

tunl1 Link encap:IPIP Tunnel HWaddr
inet addr:128.198.61.51 P-t-P:128.198.61.51 Mask:255.255.255.255
UP POINTOPOINT RUNNING NOARP MTU:1480 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)

[root@client root]# ip link show
1: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100 link/ether
00:0c:29:47:59:30 brd ff:ff:ff:ff:ff:ff
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop link/ipip 0.0.0.0 brd 0.0.0.0
3: tunl1@eth0: <POINTOPOINT,NOARP,UP> mtu 1480 qdisc noqueue link/ipip 0.0.0.0 peer
128.198.60.42

[root@client root]# route -n
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
128.198.60.42 128.198.61.1 255.255.255.255 UGH 0 0 0 tunl1
128.198.61.0 0.0.0.0 255.255.255.192 U 0 0 0 eth0
0.0.0.0 128.198.61.1 0.0.0.0 UG 0 0 0 eth0
```

You should be able to ping the target server from the client machine, and vice verse, by passing ICMP messages through IP tunnels. Run "netstat -i" on target or client or proxy machine several times during the ping session, you will see IP packages passing through tunnel interfaces.

```
[root@client root]# ping ****(target IP)
```

```
[root@proxy root]# netstat -i
Kernel Interface table
Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0 1500 0 76651 0 0 0 863 0 0 0 BMRU
tunl1 1480 0 0 0 0 0 0 0 0 0 OPRU

[root@proxy root]# netstat -i
Kernel Interface table
Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0 1500 0 76676 0 0 0 878 0 0 0 BMRU
tunl1 1480 0 52 0 0 0 52 0 0 0 OPRU

[root@proxy root]# netstat -i
Kernel Interface table
Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0 1500 0 76676 0 0 0 878 0 0 0 BMRU
tunl1 1480 0 104 0 0 0 104 0 0 0 OPRU
```

For more information on IP tunnel, please refer to: http://cs.uccs.edu/~scold/iptunnel.htm

A3 Set up SCOLD Daemon with SSL Support

A SCOLD Daemon server process named "scoldd" is set up to run on client, proxy, gateway and target machine, listening to a designated port (5115 by default), waiting for messages from the SCOLD coordinator, and setting up IPIP tunnels automatically upon requests. The communication among the coordinator, client, proxy, gateway and target is mutually authenticated and SSL encrypted.

Below is a brief summary on the compilation and usage of the SCOLD Daemon.

Compilation

Download scold daemon source code from http://cs.uccs.edu/~scold/src/scoldd/,
In scoldd directory, run

```
make scoldd
```

Running

In scoldd directory, run

```
./scoldd
```

to stop running, use the following command

```
kill -9 $(pidof scoldd)
```

Testing

For example, the coordinator wants to talk to client 128.198.61.51 on port 5111, issue the following commands

```
openssl s_client -connect 128.198.61.51:5111 -showcerts -cert certificate/ctestssl/cert/clicert.pem -key certificate/ctestssl/private/private.key -CAfile certificate/ctestssl/ca/cacert.pem
```

You can use the existing certificate in our source code when perform the above test. However, if they are expired, or you need to create your own certificate, please follow the following steps.

Creating certificate

On client, proxy, gateway, target machine,
go to certificate directory,
run "test.sh", input information as required
the testssl contain server certificate and private key
the ctestssl contain client certificate and private key
copy testssl to server testssl directory
copy ctestssl to client ctestssl directory

Verify IPIP tunnel
----
run "ifconfig" or "ip link show", should see IP tunnel configuration.
run "netstat -i" several times, should see IP tunnel traffic.
run "traceroute" before and after IP tunneling. After IP tunneling, you will not be able to see IP hops because they are replaced with * signs.
run "wget http://128.198.60.201" for web access, "ssh -l root 128.198.60.201" for ssh
"ifconfig tunl1 down" to shut down the tunnel

The source code and related configuration script of SCOLD daemon is available under:

http://cs.uccs.edu/~scold/src/scoldd/

A4 Resolve Library

The resolve library on the client machine is enhanced to support indirect routing.

In Redhat Linux, the resolve library is usually located in /usr/lib or /lib directory, and named as libresolv-nnn.so (nnn is the version). The source code of resolve library can be obtained from glibc package. The glibc package can be obtained from:

http://directory.fsf.org/GNU/glibc.html.

We modify the res_query.c file under glibc/resolv directory (version 2.3.2). To our experiences, the version of resolve library is independent of the Redhat Linux version. We have successfully deployed the v.2.3.2 glibc resolve library in several Redhat Linux versions (from 9 to Fedora core 2).

The resolve library is a shared library on Linux. The compilation of resolve library is very different from the ordinary c complication because there are many compilation parameters need to be specified. A mal-compiled resolve library can be devastating because it can easily crash the whole Linux system. For more information on shared library, please refer to:

http://www.tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html

A simpler and safer solution for compiling the resolve library is to compile the whole glibc package. Please refer to glibc complication help on how to compile the glic package. The URL of glibc help page is at:

http://www.gnu.org/software/libc/manual/html_mono/libc.html

The compilation of glibc package usually takes 15 - 60 minutes depending on the machine setting.

The compilation output of resolve library is in glibc/compile/resolv directory, and named as libresolv.so. This shared library file (.so) is what we want for the enhanced resolve library.

Run “ls -la /lib/libresolv*” to list the existing resolve library. Below is an example.

```
-rwxr-xr-x     1 root    root      73640 Nov  5  2003 /lib/libresolv-2.3.2.so
lrwxrwxrwx     1 root    root         25 Jan 30 23:30 /lib/libresolv.so.2 -> libresolv-2.3.3.so
```

Backup the existing /lib/libresolv-2.3.2.so file, and copy the new libresolv.so from glibc/compile/resolv to /lib to overwrite the existing file. Re-link /lib/libresolv.so.2 to new file if necessary.

```
-rwxr-xr-x     1 root    root      64844 Nov  5  2003 /lib/libresolv-2.3.2.so
-rwxr-xr-x     1 root    root      73640 Jul 25  2003 /lib/libresolv-2.3.3.so_backup
lrwxrwxrwx     1 root    root         25 Jan 30 23:30 /lib/libresolv.so.2 -> libresolv-2.3.3.so
```

Reboot the machine.

Now you should get the new resolve library up and running.

Run a simple command “ping www.yahoo.com”, the new resolve library will be called.

Be very careful when working on the resolve library, you can easily crash the system. It is better to do the resolve library development on a virtual machine, then migrate the compiled library to the machine you want.

If you feel regret and decide to go back to the original resolve library, just copy the original resolve library back to replace the enhanced one (assuming you can still boot up the Linux system).

The source code of resolve library is available under:

http://cs.uccs.edu/~scold/src/glibc/

A5 Enhanced DNS Bind9

For more information on nsreroute and enhanced DNS, please refer to David Wilkson's master thesis:

http://cs.uccs.edu/~chow/pub/master/dbwilkin/doc/dwilkinson_thesis.doc

The source code can be obtained at:

http://cs.uccs.edu/~chow/pub/master/dbwilkin/src

A6 A step by step demo

Below is a step by step SCOLD demo. It is based on the tested shown in Figure A2.

Testbed setup:

boot up the virtual machines and login to each machine

username: root

password: ****(available upon request, contact me at caiyu_usa@yahoo.com or Dr. chow at chow@cs.uccs.edu)

vmware file location:

    ardor.uccs.edu:

    d:/vmware/ycai2,

    client, proxy, altgw, maingw, target 5 directories.

make sure the network cards of vmware is correctly configured: use bridged virtual network connection for eth0, use host-only virtual network connection for the rest network interfaces.

To save disk space,

   run clientDNS on the same client machine,

   run targetDNS, the coordinator on the same target machine,

   run DDoS attackers on the same main gateway machine,

Demo steps:

1) Open 5 vmware machines: client, proxy, altgw, maingw, target,

2) Go to "/home/ycai/sslres" directory on all those machines

3) Run script "sh init.sh" on all those machines to initialize the machines, including procedures like run the scold demo, set up direct route, set the routing table.

4) Now to show the direct route:

on the client machine, you can do the followings:

ping 192.168.4.2(target.csnet.uccs.edu) from 192.168.0.1(client.csnet.uccs.edu)

verify the direct route by "traceroute target.csnet.uccs.edu", 2 hops

run "sh http_demo.sh" to see the http download of a big file with speed average of 50k - 60k/s from the target sever.

5) Now launch DDoS attack

run "sh ddos_attack.sh" on the target machine to launch the attack,

run "sh http_demo.sh" on the client machine to see the http download speed drop dramatically to 1k - 10k/s

6) Now start the indirect route

(stop the "sh http_demo.sh" job on client machine in step 5)

run "sh indirec_route.sh" on the target machine to launch the indirect route,

run "sh http_demo.sh" on client machine to see the indirect route, you will see the initil setup delay. But after that, it runs fast (about 40k/s)

verify the indirect route by "traceroute target.csnet.uccs.edu"

7) If needed, run "cleanipip.sh" to clean up the indirect route.

Below are references on installation for enhanced secure dns (sdns) and resolve library.

sdns installation:

1) get sdns source file from gandalf.uccs.edu, get bind source from Internet

2) get openssl file from gandalf: /usr/include/openssl

3) compile:

"./configure -with-openssl"

"make"

"make depend"

"make install"

4) get zone file from gandalf:/var/named

5) get bind conf file from gandalf: /etc/named.conf

libresolve installation:

1) source file in athena.uccs.edu:~ycai/glibc/resolv/res_query.c

2) go to glibc/compile, and run make to compile

3) go to glibc/compile/resolv and look for libresolv.so

4) copy the libresolv.so to the client machine /lib directory,

"ls -la /lib/libresolv*"

"rm -f /lib/libresolv.so.2"

"ln -s /lib/libresolv.so /lib/libresolv.so.2"

APPENDIX B

TCP CONGESTION CONTROL AND LINUX KERNEL

B1 TCP Congestion Control

TCP is an end to end transport layer protocol which operates over the heterogeneous Internet. TCP has no prior knowledge of the network characteristics, thus it has to adjust its behavior according to the returned ack message and know the current state of the network. TCP has built-in support for congestion control, which ensures that TCP does not pump too many data packets than what the network can handle. TCP is a complex protocol; hence we only introduce aspects related to TCP congestion control in this appendix.

Figure B1 shows the evolution of TCP versions. After the first Internet congestion collapse in 1986, Van Jacobson proposed the TCP congestion control mechanism.

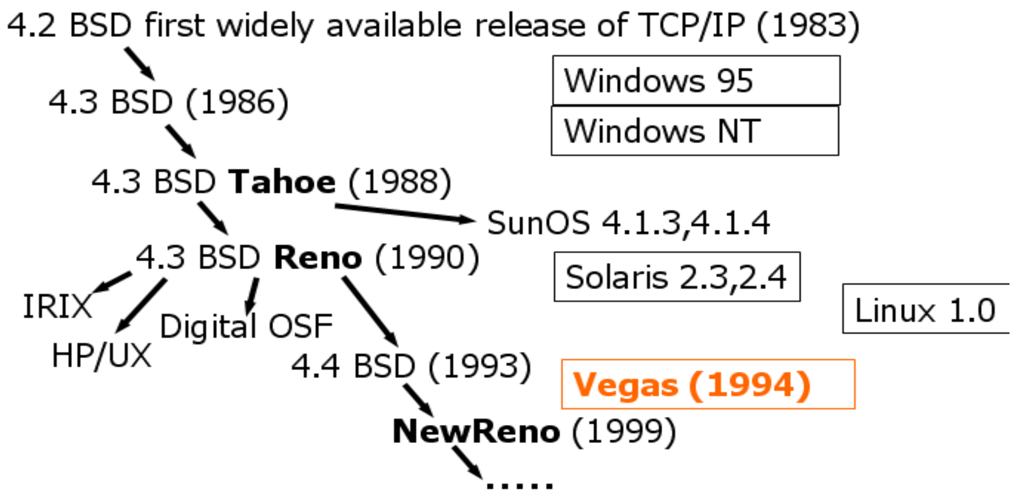


Figure B1: TCP version [SLow02]

In a TCP session, each byte has a sequence number, and ACKs are cumulative.

Sliding window

TCP supports windowing scheme — the process of sending data packets in sequence without waiting for an intervening acknowledgement. The sliding window in TCP serves several purposes. (1) It guarantees the reliable delivery of data. (2) It ensures that the data is delivered in order. (3) It enforces flow control between the sender and the receiver. See Figure B2 below for TCP sliding window.

awnd = MaxRcvBuffer - (LastByteRcvd - NextByteRead)

flightsize = min(awnd, cwnd)

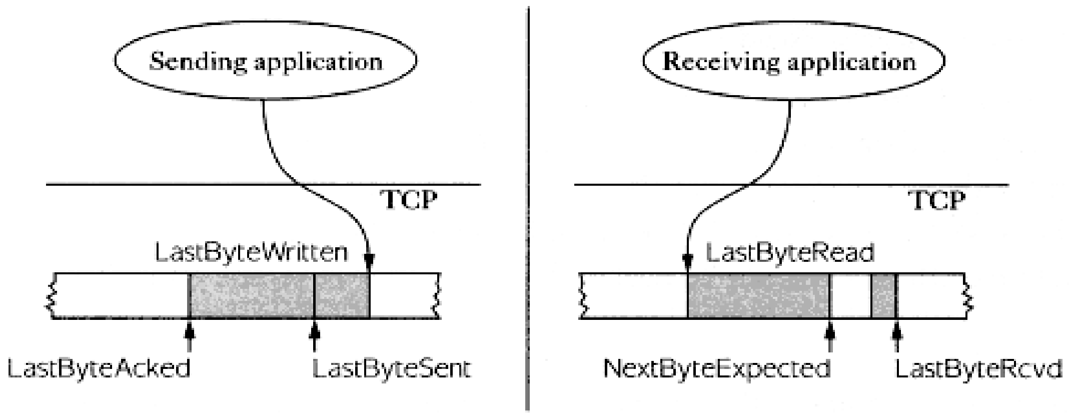


Figure B2: Sliding window in TCP [SSin]

TCP Congestion Control

TCP flow control is based on the premise that an out-of-order packet is an indication of packet loss. Note that this may not be true in a multipath environment since the out-of-order packets may be in transmit over other paths. Packet loss is detected by Retransmission Time-Out (RTO timer) or Duplicate ACKs (usually 3).

- When Time-out occurs, TCP enters slow start.
- When dup ACKs occurs, TCP enters fast retransmit and fast recovery.

TCP has four defined congestion control mechanisms to ensure the most efficient use of bandwidth, and quick error and congestion recovery. The four mechanisms, defined in detail in RFC 2581, are:

– Slow Start – Congestion Avoidance

– Fast Retransmit – Fast Recovery

Retransmission Timeout

The first error-detection and error-recovery mechanism is the retransmission timer. The value specified by this timer is referred to as the retransmission timeout (RTO). When RTO, TCP cuts congestion window in half and enter slow start.

- ssthresh ← cwnd/2
- cwnd= 1

Slow Start

As the name suggests, "Slow Start" starts slowly, increasing its window size as it gains confidence about the networks throughput. A TCP connection starts in the "Slow Start" state. In this state, TCP adjusts its transmission rate based on the rate at which the acknowledgements are received from the other end.

TCP Slow start is implemented using two variables, cwnd (Congestion Window) and ssthresh (Slow Start Threshold). cwnd is a self imposed transmit window restriction at the sender end. cwnd will increase as TCP gains more confidence on the networks ability to handle traffic. ssthresh is the threshold for determining the point at which TCP exits slow start. If cwnd increases beyond ssthresh, the TCP session in that direction is considered to be out of slow start phase. Figure B3 shows the TCP slow start.

- Start with cwnd = 1 (slow start)
- On each successful ACK, increment cwnd, cwnd ← cwnd + 1
- Exponential growth of cwnd, for each RTT, cwnd ← 2 * cwnd
- Enter CA when cwnd >= ssthresh

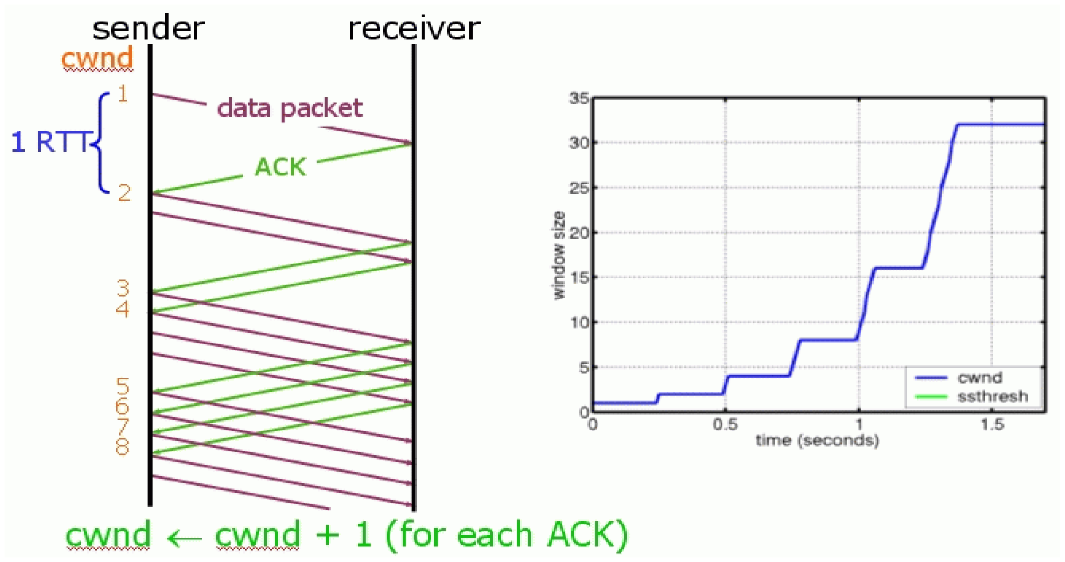


Figure B3. TCP slow start [Slow02]

Congestion Avoidance

When cwnd exceed ssthresh, TCP will be out of slow start and enter congestion avoidance. Exiting slow start signifies that the TCP connection has reached an equilibrium state where the congestion window closely matches the networks capacity. From this point on, the congestion window will not move geometrically. cwnd will move linearly once the connection is out of slow start.

- Starts when cwnd >= ssthresh
- On each successful ACK: cwnd ← cwnd + 1/cwnd
- Linear growth of cwnd, for each RTT: cwnd ← cwnd + 1

Fast Retransmit and Fast Recovery

TCP Fast Retransmit and Fast Recovery have been designed to speed up the recovery of the connection when packet loss occurs. Fast Retransmit and Recovery detect packet loss via duplicate acknowledgements. When a packet segment is lost, TCP at the receiver will keep sending ack segments indicating the next expected sequence number. This sequence number would correspond to the lost segment. If only one segment is lost, TCP will keep generating acks for the following segments. This will result in the transmitter getting duplicate acks, acks with the same ack sequence number.

Fast Retransmit: TCP receives duplicate acks and it decides to retransmit the segment, without waiting for the segment timer to expire. This speeds up the recovery of the lost segment.

Fast Recovery: Once the lost segment has been transmitted, TCP tries to maintain the current data flow by not going back to slow start. TCP also adjusts the window for all segments that have been buffered by the receiver.

The fast retransmit and fast recovery algorithms are usually implemented together as follows.

1. When the third duplicate ACK in a row is received, set ssthresh to one-half the current congestion window, cwnd, but no less than two segments. Retransmit the missing segment. Set cwnd to ssthresh plus 3 times the segment size. This inflates the congestion window by the number of segments that have left the network and which the other end has cached.

2. Each time another duplicate ACK arrives, increment cwnd by the segment size. This inflates the congestion window for the additional segment that has left the network. Transmit a packet, if allowed by the new value of cwnd.

3. When the next ACK arrives that acknowledges new data, set cwnd to ssthresh (the value set in step 1). This ACK should be the acknowledgment of the retransmission from step 1, one round-trip time after the retransmission. Additionally, this ACK should acknowledge all the intermediate segments sent between the lost packet and the receipt of the first duplicate ACK. This step is congestion avoidance, since TCP is down to one-half the rate it was at when the packet was lost. Figure B4 shows the congestion control of TCP reno. Figure B5 shows its fast retransmit.

Enter Fast Retransmit and Fast Recovery after 3 dup ACKs

- Set ssthresh $\leftarrow$ cwnd/2
- Retransmit lost packet
- Set cwnd $\leftarrow$ ssthresh + 3 (window inflation)
- For each successive duplicate Ack: Increment cwnd by 1; New packets are transmitted if allowed by cwnd
- On non-dup ACK (1 RTT later), set cwnd $\leftarrow$ ssthresh (window deflation)
- Enter CA

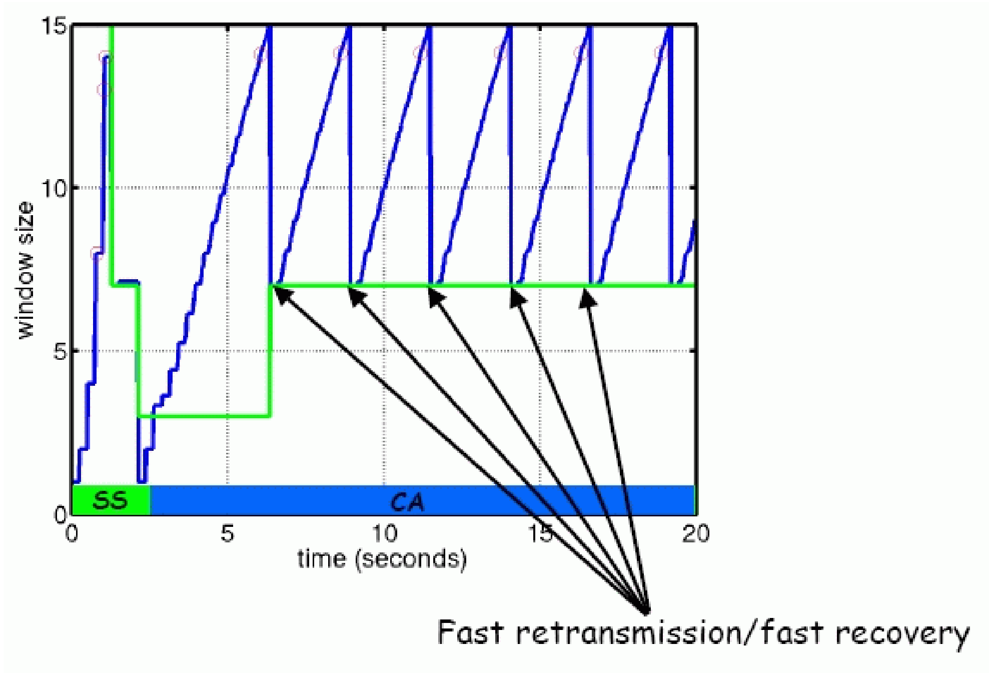


Figure B4: TCP Reno Fast Retransmission and Fast Recovery [Slow02]

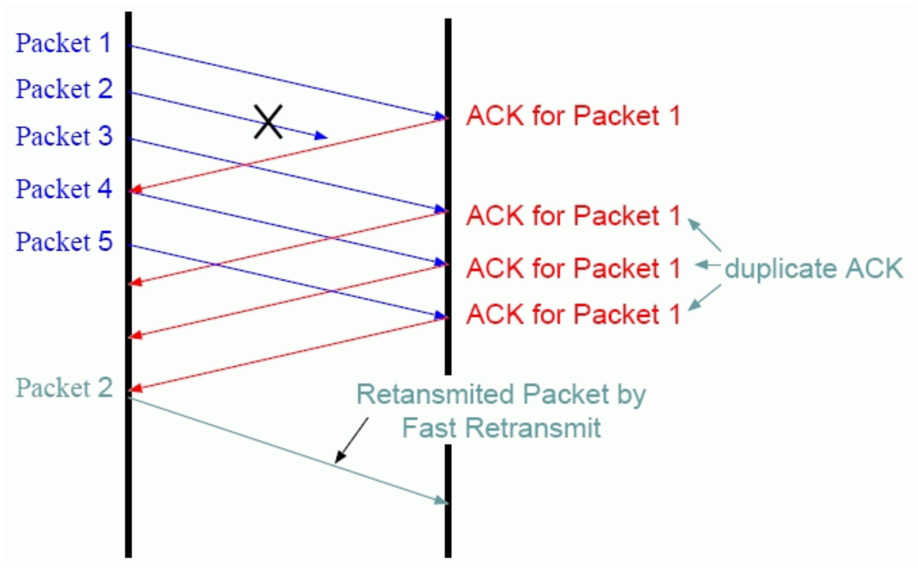


Figure B5: TCP fast retransmit [Slow02]

SACK

Cumulative ACK style in TCP is ambiguous, when multiple packets are lost. Selective Acknowledgment (SACK) provides more precise information about packet. SACK is defined in RFC2018. Figure B6 shows an example using SACK.

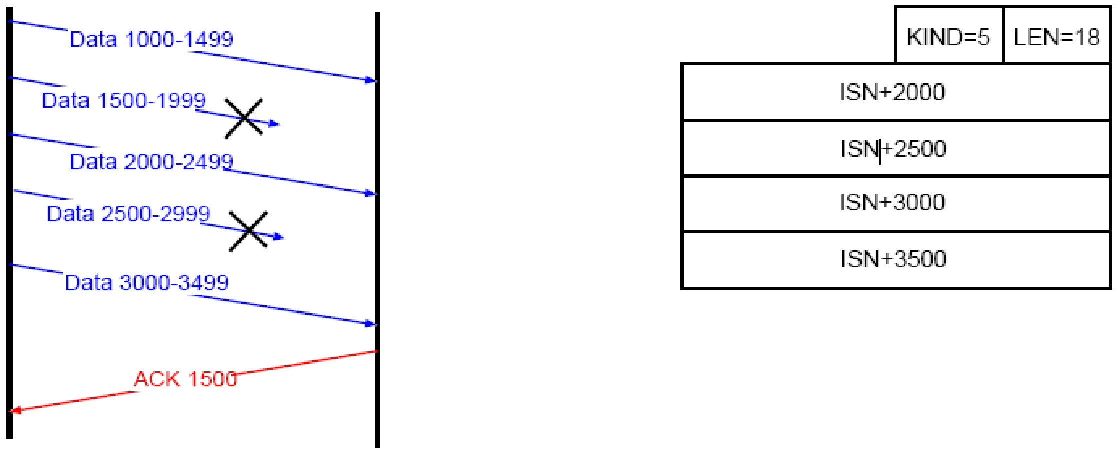


Figure B6: SACK [Slow02]

B2 TCP/IP in Linux Kernel

In Linux kernel TCP/IP implementation, packets are stored in struct sk_buffs. And kernel-side correspondent for TCP socket is struct sock. Most network code that we are going to modify in Linux kernel is located in linux/net/ipv4 directory.

struct sock holds state data for the socket (such as the TCP variables regarding congestion window, etc.). There are several queue pointers: outgoing packets not yet acknowledged, incoming packets not yet delivered to application. Queues hold chains of sk_buffs. sk_buff usually corresponds to one packet sent/received to network. In addition to packet data, there are protocol headers and control information in sk_buff. Figure B7 shows the structures. In our PSMC, we are going to add one more receive queue as double buffer.

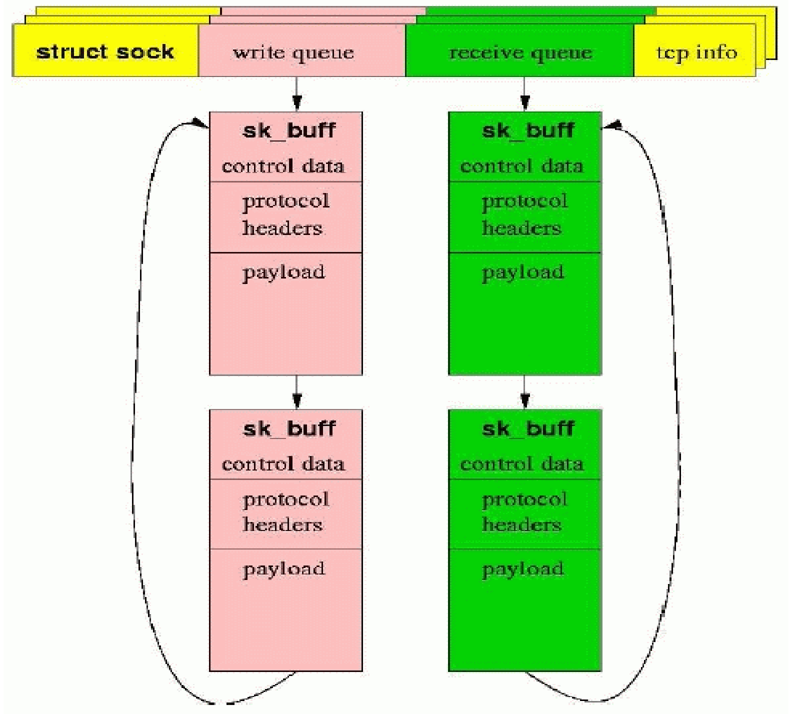


Figure B7: sock and sk_buff [PSar02]

Figure B8 shows the packet handling in Linux kernel.

Linux kernel programming is very different from user-level programming. Usage of virtual machine is strongly advised for kernel development.

For more details on Linux kernel and TCP/IP, please refer to [GHer, HWel].

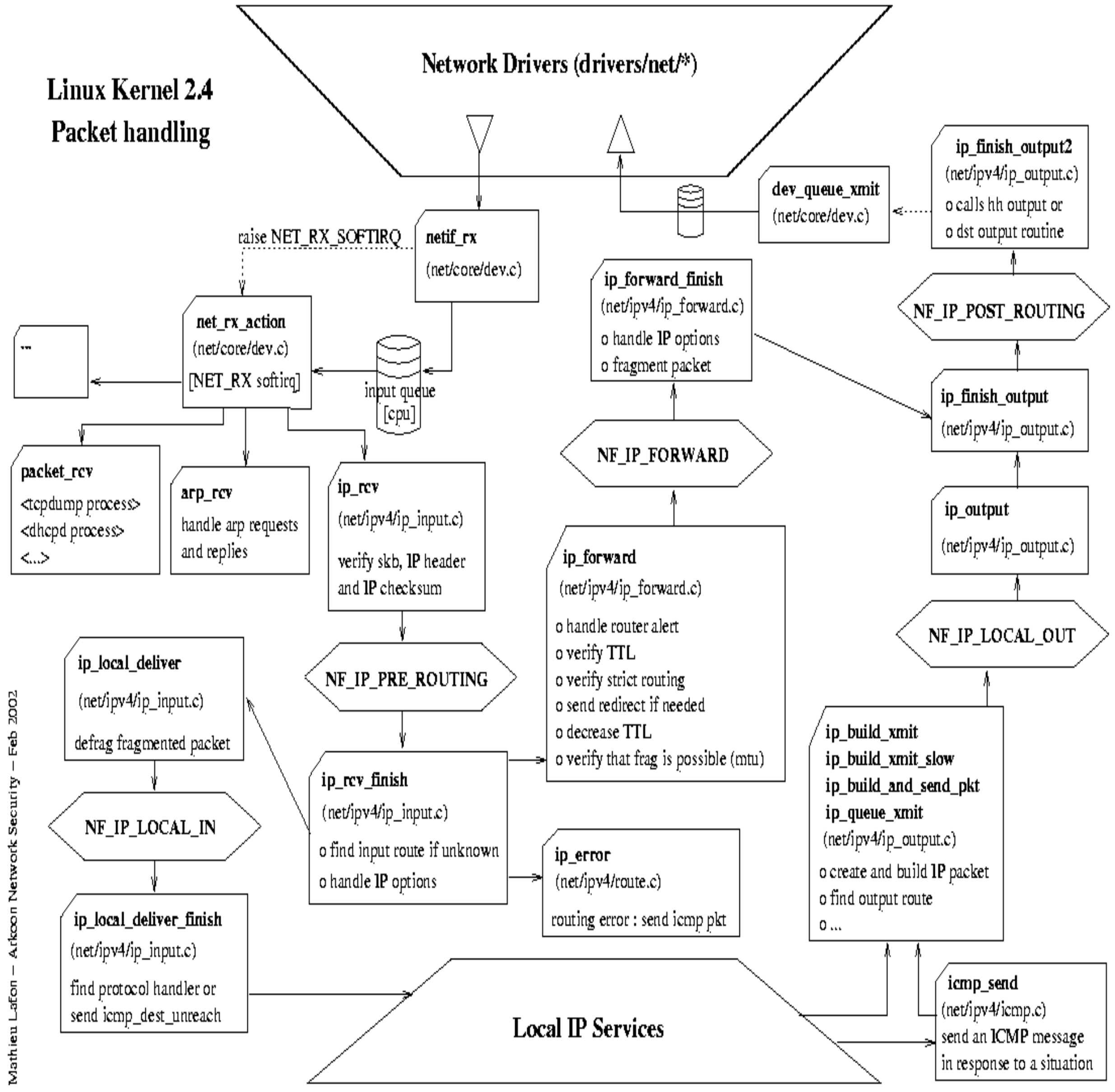

Linux Kernel 2.4
Packet handling
Network Drivers (drivers/net/*)
netif_rx (net/core/dev.c)
raise NET_RX_SOFTIRQ
net_rx_action (net/core/dev.c) [NET_RX softirq]
input queue [cpu]
...
packet_rcv <tcpdump process> <dhcpd process> <...>
arp_rcv handle arp requests and replies
ip_rcv (net/ipv4/ip_input.c) verify skb, IP header and IP checksum
NF_IP_PRE_ROUTING
ip_rcv_finish (net/ipv4/ip_input.c) o find input route if unknown o handle IP options
ip_local_deliver (net/ipv4/ip_input.c) defrag fragmented packet
NF_IP_LOCAL_IN
ip_local_deliver_finish (net/ipv4/ip_input.c) find protocol handler or send icmp_dest_unreach
ip_error (net/ipv4/route.c) routing error : send icmp pkt
ip_forward (net/ipv4/ip_forward.c) o handle router alert o verify TTL o verify strict routing o send redirect if needed o decrease TTL o verify that frag is possible (mtu)
NF_IP_FORWARD
ip_forward_finish (net/ipv4/ip_forward.c) o handle IP options o fragment packet
dev_queue_xmit (net/core/dev.c)
ip_finish_output2 (net/ipv4/ip_output.c) o calls hh output or o dst output routine
NF_IP_POST_ROUTING
ip_finish_output (net/ipv4/ip_output.c)
ip_output (net/ipv4/ip_output.c)
NF_IP_LOCAL_OUT
ip_build_xmit ip_build_xmit_slow ip_build_and_send_pkt ip_queue_xmit (net/ipv4/ip_output.c) o create and build IP packet o find output route o ...
Local IP Services
icmp_send (net/ipv4/icmp.c) send an ICMP message in response to a situation
Mathieu Lafon – Arkoon Network Security – Feb 2002

Figure B8: Packet handling in Linux kernel [GHer]

APPENDIX C

PSMC USER MANUAL

This user manual explains the steps required to setup, configure and run the proxy server based multipath connection (PSMC) network for demonstrating the enhanced TCP/IP protocol with multiple paths. Sample demonstration scripts and files are provided as a step-by-step guide on how PSMC works. The set up of the PSMC testbed includes three main components:

- The sender network
- The receiver network
- The proxy servers

C1 PSMC

We have set up several different PSMC testbeds for different research purposes. Figure C1 is a PSMC testbed on real machines. It is more realistic to measure the multipath performance on real machines. Figure A1 in Appendix A is another PSMC testbed based on VMWare virtual machines. It is more convenient to do Linux kernel development on virtual machines.

1) Set up proxy server based overlay network

We first need to set up the proxy server based overlay network. The procedure is the same as setting up SCOLD testbed. Please refer to Appendix A for more information.

2) Set up Linux kernel development environment

User Mode Linux (UML) can be set up for Linux kernel development. For more information on UML, please refer to Frank Watson’s master thesis [Fran05].

For Linux kernel compilation, please refer to the related documents. Note that different kernel version may have slightly different compilation steps.

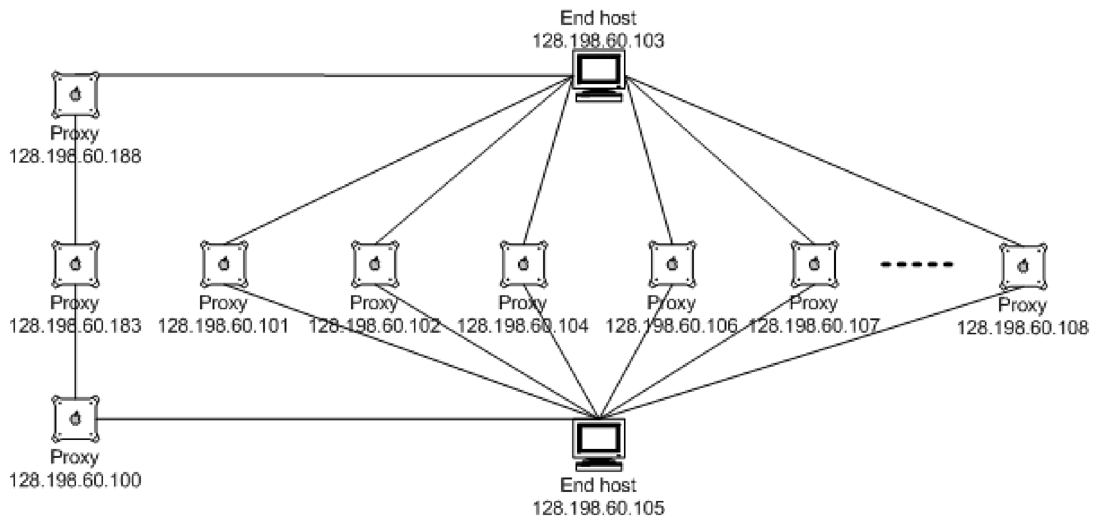


Figure C1: PSMC testbed

The command printk() can be used to print kernel debug information during the kernel code execution.

Source insight is a good source code navigation tool. It has Linux and Windows version. Linux Cross Referencing web site (http://lxr.linux.no/) can also be used for reading source code. For Linux kernel programming, reading and understanding the existing code is the first and important step. From my experience, if you know where to modify the code and what the code does, then you almost finish 80% of the job.

During kernel programming, the modified kernel may crash from time to time. Therefore, set up a development platform on virtual machine is very helpful. You can use UML, VMWare or Virtual PC to set up the virtual machine development environment.

3) IP striping

PSMC packet striping is implemented on the IP layer. We modify the ip_output.c file under the linux/net/ipv4 directory. To modulate the code, we put the PSMC functional code in a PSMC module, and insert a function pointer in the ip_output.c. Therefore, we can easily load, unload and update the PSMC code without recompiling the whole kernel. We implemented a weighted round robin packet striping scheme by now. More data striping schemes can be implemented on the PSMC module without modifying the Linux kernel again.

4) Double buffering

We first redefine the struct sock by adding a scoldLog struct, which is similar to backlog. A sock struct is correspondent to a socket connection; a sk_buff struct is correspondent to a packet in the socket connection. And the new scoldLog is correspondent to the double buffer which temporarily holds the packets between IP and TCP. The scoldLog is a packet link list (sk_buff link list) like backlog. Since we don't physically copy and store the packets in double buffer, the overhead of double buffer is minimized.

Then we modify the tcp_ipv4.c file under linux/net/ipv4 directory to implement the double buffer algorithm. Right now we implemented an adaptive double buffer algorithm by dynamically changing the buffer size according to the network condition. However, the double buffer imposes noticeable queueing delay on packets, therefore a more efficient algorithm needs to be designed.

For more information on the PSMC code implementation, please refer to:

http://cs.uccs.edu/~chow/pub/master/ycai/src/psmc

5) PSMC Daemon

There are PSMC management daemons running end hosts. The primary responsibilities of PSMC daemon include the followings: update the psmc_wr file, analyze the traffic information, dynamically adjust the packet stripping ratio, dynamically adjust the double buffer size, exchange traffic information with the other end. PSMC daemon analyzes the traffic information by running tcpdump, which imposes noticeable overhead. A more efficient monitoring approach needs to be designed and implemented.

The psmc_wr file stores the destination, weight, proxy server IPs and tunnel device information. It is updated by SCOLD daemon and PSMC daemon. A sample of psmc_wr file is as follow.

```
#destination IP: proxy IP: device name: striping ratio

128.198.60.105:128.198.60.102:tunl1:2

128.198.60.105:128.198.60.104:tunl2:5

128.198.61.51:128.198.60.104:tunl2:1

128.198.61.51:128.198.60.106:tunl4:1
```

6) Proc file system

The Linux proc file system can be used to communicate information between the kernel and the user to tune the kernel performance. The proc system is a virtual file system. The PSMC uses the proc file system to turn off and on the multiple path routing and control other variables used for PSMC logging and buffering. A sample usage is like bleow.

```
root@client:/proc/sys/net/ipv4/multipath# cat bufferOn

0

root@client:/proc/sys/net/ipv4/multipath# echo "1" > bufferOn

root@client:/proc/sys/net/ipv4/multipath# cat bufferOn

1
```

7) Packet dropper and packet delayer

In PSMC performance evaluation, we need to adjust the lost rate and latency on certain routes to test different scenarios. This is accomplished by using a packet dropper and packet delayer on the proxy servers. We modify the dev_queue_xmit() function in /linux/net/core/dev.c file by periodically free skb (drop packet) or wait for a period (add latency). The end user can specify the parameters like drop rate or latency via proc file system. A sample code is listed below.

```
// modify dev_queue_xmit() function in /linux/net/core/dev.c for packet dropper

int dev_queue_xmit (struct sk_buff *skb){

…

   if (drop_packet()) {

        kfree_skb(skb);

   }

….

}
```

8) Rate limiting

In PSMC performance evaluation, we need to adjust the bandwidth on selected routes. This can be accomplished by using rate limiting on proxy server. Below is a scrip example which limits 50 packets per second.

```
iptables –flush

rate=50

iptables -A OUTPUT -p tcp -m limit --limit $rate/second -j ACCEPT

iptables -A OUTPUT -p tcp -j DROP

iptables -A OUTPUT -p udp -m limit --limit $rate/second -j ACCEPT

iptables -A OUTPUT -p udp -j DROP
```

We can use iptraf to monitor bandwidth and traffic condition on proxy server.

C2 A step by step demo

We use the PSMC testbed illustrated in Figure c2. The two end hosts install the PSMC modules, the proxy servers install the packet dropper, packet delayer and bandwidth rate limiting.

1) Run ipip_init.sh on client, proxy, server machines to set up the overlay network and impose rate limiting.

2) Run “wget server” from client machine to download a large file and monitoring the bandwidth (5Mb/s). By using “ethereal” on client, we can monitor the traffic condition and see how the packets are transmitted. This is for single path scenarios.

3) Run psmc_init.sh on client and server machines to turn on the multipath and set double buffer size.

4) Run “wget server” from client machine to download a large file and monitoring the bandwidth (10Mb/s). By using “ethereal” on client, we can monitor the traffic condition and see how the packets are transmitted via two paths. This is for two-path scenarios.

5) Run add_proxy.sh on client machine, a new indirect route will be set up.

6) Run “wget server” from client machine to download a large file and monitoring the bandwidth (15Mb/s). By using “ethereal” on client, we can monitor the traffic condition and see how the packets are transmitted via three paths. This is for three-path scenarios.