2006-04-06

Exploiting Flow Relationships to Improve the Performance of Distributed Applications

Hao Shang
Worcester Polytechnic Institute

Exploiting Flow Relationships to Improve the Performance of Distributed Applications

by

Hao Shang

A Dissertation

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

in

Computer Science

by

January 1, 2006

APPROVED:

Prof. Craig E. Wills Advisor	Prof. Mark Claypool Committee Member
Prof. Robert E. Kinicki Committee Member	Dr. Ralph Droms Cisco Systems External Committee Member
Prof. Michael Gennert Head of Department	

Abstract

Application performance continues to be an issue even with increased Internet bandwidth. There are many reasons for poor application performance including unpredictable network conditions, long round trip times, inadequate transmission mechanisms, or less than optimal application designs. In this work, we propose to exploit flow relationships as a general means to improve Internet application performance. We define a relationship to exist between two flows if the flows exhibit temporal proximity within the same scope, where a scope may either be between two hosts or between two clusters of hosts. Temporal proximity can either be in parallel or near-term sequential.

As part of this work, we first observe that flow relationships are plentiful and they can be exploited to improve application performance. Second, we establish a framework on possible techniques to exploit flow relationships. In this framework, we summarize the improvements that can be brought by these techniques into several types and also use a taxonomy to break Internet applications into different categories based on their traffic characteristics and performance concerns. This approach allows us to

investigate how a technique helps a group of applications rather than a particular one. Finally, we investigate several specific techniques under the framework and use them to illustrate how flow relationships are exploited to achieve a variety of improvements.

We propose and evaluate a list of techniques including piggybacking related domain names, data piggybacking, enhanced TCP ACKs, packet aggregation, and critical packet piggybacking. We use them as examples to show how particular flow relationships can be used to improve applications in different ways such as reducing round trips, providing better quality of information, reducing the total number of packets, and avoiding timeouts.

Results show that the technique of piggybacking related domain names can significantly reduce local cache misses and also reduce the same number of domain name messages. The data piggybacking technique can provide packet-efficient throughput in the reverse direction of a TCP connection without sacrificing forward throughput. The enhanced ACK approach provides more detailed and complete information about the state of the forward direction that could be used by a TCP implementation to obtain better throughput under different network conditions. Results for packet aggregation show only a marginal gain of packet savings due to the current traffic patterns. Finally, results for critical packet piggybacking demonstrate a big potential in using related flows to send duplicate copies to protect performance-critical packets from loss.

Acknowledgments

I would like to express my gratitude to my advisor, Prof. Craig E. Wills, for his support, advice, patience, and encouragement throughout my graduate studies. It is not often that one finds an advisor and colleague that always finds the time for listening to the little problems and hurdles that unavoidably crop up in the course of performing research. His technical and editorial advice was essential to the completion of this dissertation and has taught me innumerable lessons and insights on the workings of academic research in general.

My thanks also go to the members of my Ph.D. committee, Prof. Robert E. Kinicki, Prof. Mark Claypool, and Dr. Ralph Droms, who provided valuable feedback and suggestions to my previous talks and dissertation drafts that helped to improve the presentation and contents of this dissertation.

The friendship of Chunling Ma, Mingzhe Li, Jae Chung, Huahui Wu, Feng Li, and all the other previous and current PEDS members is much appreciated. They have contributed to many interesting and good-spirited discussions related to this research. They also provided tremendous mental support to me when I got frustrated at times.

Last, but not least, I would like to thank my wife Lily for her understanding and love during the past few years. Her support and encouragement was in the end what made this dissertation possible. My parents receive my deepest gratitude and love for their dedication and the many years of support during my studies.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet has grown from its original four hosts in 1969 to currently millions of hosts, from a single Email application to currently tens of popular applications. It has become one of the most important communication mechanisms for information dissemination and individual interactions without regard to geographic location. Many research topics have focused on the improvement of Internet application performance. We have selected one interesting direction by exploring and seeking to exploit flow relationships.

1.1 Application Performance Is Still An Issue

The speed of the Internet has substantially improved from the time when it was invented, which brings opportunities as well as challenges. Both the variety and instances of applications have significantly increased. Many applications built for local area networks (LANs) have been extended to be

used in wide area networks (WANs). However, end users are still experiencing unsatisfactory Internet performance due to many reasons including the intermediate network, transport protocols, and applications themselves.

Internet resources are still limited and network conditions change dramatically due to Internet traffic dynamics. The best effort behavior of the current Internet does not guarantee an adequate network environment for all applications all the time. Improving application performance under poor network conditions is still necessary.

Even with ample network bandwidth, applications may still perform poorly due to inherent latency between two end hosts. For example, the three-way handshake procedure required by the TCP connection establishment introduces one round trip time (RTT) delay before any user data can be exchanged, which causes a performance problem for long RTT paths. A badly designed application may also incur unnecessary packet exchanges. One example is that the sequential transmission scheme used by old Web browsers establishes a TCP connection for each Web object. Studies on how to accelerate transaction procedures are still needed.

The widely used TCP and UDP transmission protocols are not always suitable in terms of application performance. On one hand, TCP enforces additive-increase/multiplicative-decrease (AIMD) congestion control as well as reliable in-order transmissions. On the other hand, UDP does not do any control except providing multiplexing over IP. Applications may find that neither protocol performs optimally. UDP is not responsive to any network congestion and the throughput of a TCP connection may be much less than

available bandwidth (e.g. for a long RTT, but high bandwidth path). Optimizing existing transmission schemes or building new transmission protocols are also required.

With new applications coming into use, application performance concerns become more diverse. For example, "telnet" or "ssh" are resilient to available bandwidth but sensitive to long RTTs, while applications such as non-interactive video or audio streaming (e.g. video/audio on demand) can tolerate long RTTs but perform poorly if required minimal available bandwidth is not met. Providing satisfactory performance for a variety of applications is challenging.

Consequently, improving application performance is an important Internet-related research topic. Researchers have taken approaches from different directions. Proposals like integrated [BCS94] or differentiated [BBC+98] services seek to provide enhanced network services for data flows or groups of flows. These approaches need participation of all intermediate routers between two end hosts. Numerous studies have sought to achieve better performance by improving transmission mechanisms under various network conditions. As an example, [JBB92] tries to improve the performance of TCP for large bandwidth/delay product paths by using scaled windows and timestamp options. Another set of studies, such as stream control transmission protocol [SXM+00] and T/TCP [Bra94], seek to design new transmission protocols for particular application needs. Much work has also been done at the application level. For example, persistent connection and pipelining mechanisms are introduced in HTTP/1.1 [FGM+99] to improve the performance of Web applications.

While the above approaches seek to improve application performance from various angles, two problems still exist. First, many approaches propose to improve transmission throughput or provide prioritized services. However, even with sufficient bandwidth, providing satisfactory application performance is not guaranteed. Long round trip times and inefficient use of available bandwidth may also cause performance problems, especially for applications that require multiple packet interactions before the user can get a response.

Second, few approaches seek to improve transmission efficiency or exploit the unused packet capacity. We use *unused packet capacity* to refer to the available packet space in non-full packets, which can be presented as the difference between the size of an IP packet and its path maximum transmission unit (MTU) size [MD90]. While MTU sizes for most link types are over 1500 bytes [1], about only 20% of packets reach this size and most packets are small as found in our background study described in Chapter 3 and many other Internet traffic studies. As the permitted MTU size grows [KSO+01, BDH99], the problem of having much unused packet capacity becomes more severe.

The described problems pose a challenge to application performance and will become more serious as the variation of network applications grows. As the "last mile" problem is being resolved by cable and ADSL, the Internet backbone may become the new bottleneck. Approaches to im-

[1] During the time of this study, we were aware of only three link types using MTU size under 1500 bytes: IEEE 802.3/802.2, PPPoE, and X.25. The first two link types use MTU size of 1492, which is very close to 1500 bytes. For convenience, we treat them the same as Ethernet, which uses MTU size of 1500 bytes. The X.25 link type uses MTU size of 576 bytes, but this link technique is rarely used nowadays.

proving application performance as well as increasing network efficiency are required. In this thesis, we address these problems by seeking to exploit relationships between network flows.

1.2 Observation of Flow Relationships

We define a network *flow* as a stream of data sharing the same end points. An end point is a virtual entry inside a host, which is normally identified as an assigned port along with its associated protocol. We define a *relationship* to exist between two flows if packets inside the flows exhibit temporal proximity within the same scope. Scope may either be between two hosts or between two clusters of hosts where a *cluster* is the set of hosts at a site sharing the same end router. Temporal proximity can be either *concurrent* where one flow has temporal overlap with another flow, or *sequential* where one flow follows another that recently terminates. By extension, a flow has a relationship with itself as packets within a flow have temporal proximity.

It is not interesting to study flow relationships if the number and types of flows are few. In 1970's and 1980's, the use of the Internet was sparse and the major applications included only E-Mail, file transfer, remote login and news. The number of network flows generated per host, even per subnet, was limited. A flow had few chances to overlap with another.

The situation began to change when a new application, the WWW (World Wide Web), was invented in the early 1990's, which brought millions of users to the Internet. While the WWW did not change any underlying facilities, this application made them easier to use. Along with the boom of the

WWW from the middle 1990's, more network applications have come into use. Applications like streaming media, peer-to-peer, grid, instant messengers, and network games are common nowadays in addition to the WWW and other traditional applications [FML+03]. Both the number and the variety of network flows have significantly increased, which brings opportunities for us to observe flow relationships.

In this thesis, we have observed the existence of flow relationships by doing a background study on several traffic logs. There are over 10K flows generated per minute on average for a regular day from WPI to the rest of the Internet. Many flows end with the same pair of hosts or clusters within a short period of time. This observation indicates a considerable amount of relationships between network flows. We elaborate on the details in Chapter 3.

While a relationship is represented by temporal proximity between flows, it may be caused by different reasons.

- Application Behavior: An application may initiate multiple flows in parallel or sequentially for the completion of a transaction. One example is a streaming application that normally generates two flows—one for control and one for data. Another example is when an application network flow is preceded by a DNS (Domain Name System) lookup causing a network flow between a local DNS server and a remote authoritative DNS server. The relationships observed in this type are caused by application themselves, no matter how users use them and what content data are involved.

- Content Relationship: Concurrent or sequential flows may be generated because the content involved in those flows has an internal relationship. An example is when multiple servers at a Web site serve content for a page leading to concurrent cluster-to-cluster network flows when a Web browser downloads the page content. This type of relationship can be deduced once the relationship among content is known.

- User Behavior: Relationships other than the above two fall into this category, which is due to user access behavior. Users in one site may access servers of another site simultaneously, which causes multiple network flows existing in parallel between cluster pairs. One possibility could be a group of local users playing on-line games by using the same remote server. A flash crowd is another example, where a number of users attempt to access a same site due to a common interest.

Flow relationships can be exploited to improve application performance. The relationships caused by particular application behavior or internal content relationships normally have relatively fixed patterns. In many cases, future flows or packets can be predicted given a known relationship. By removing predicted traffic from the critical path of an application, the number of WAN round trips is reduced, resulting in improved performance.

A relationship observed due to application behavior may also be used as an indicator of whether a particular application implementation is optimal. Web browsers prior to HTTP/1.1 used parallel TCP connections to

download multiple Web objects from the same server, which is considered inefficient in the term of resource usage. Persistent connection and pipelining requests supported by HTTP/1.1 make better use of network resources by allowing multiple objects to be transmitted over one TCP connection.

Concurrent or sequential relationships can be used for an application to piggyback any useful information with an ongoing flow. We have observed that many packets are smaller than the path MTU (Maximum Transmission Unit). The unused packet capacity can be exploited to help applications in a number of ways. Applications may be benefited from improved performance without introducing any extra transmissions.

1.3 Lack of Systematic Study on Flow Relationship

With the ever increasing instances and varieties of applications, opportunities to observe flow relationships have significantly increased. Those relationships can be exploited to improve application performance. However, little research work has been done in this area and none in a systematic way.

Most applications are developed with little knowledge of others. While the independence brings ease in the sense of development, it may not exploit relationships across applications. Consequently, much research work has focused on particular applications and tried to improve the performance of an application on its own, while little work has examined and exploited relationships across applications.

There is also a lack of coordination between applications and under-

lying transmission layers. An application has no way to convey its specific transmission requirements. Without such a mechanism, it is hard for transmission layers to coordinate flows from different applications. As a result, flow relationships may not be well exploited as transmission layers do not have any knowledge of how to handle those flows. The application level framing (ALF) principle [CT90] states that the underlying transmission mechanism should be properly designed to help applications meet their specific objectives. However, proposals to improve a particular protocol are many, while suggestions for better coordination between applications and transmissions are few.

Several studies that exploit flow relationships have limited scopes. For example, both Ensemble-TCP [EHT00] and Congestion Manager [BRS99] seek only to coordinate flows within the same host pair. However, modern applications tend to be distributed. Services involving intensive computing or a large volume of requests normally require a farm of servers instead of a single host. Related flows do not necessarily have the same end hosts. Expanding the study scope from host-to-host to cluster-to-cluster is necessary, as flows ending with the same cluster pair share a common path that is the primary contributor of transmission delay and the source of dynamic network conditions including loss, congestion and jitter. Techniques, such as information piggybacking and coordinated congestion control, may also apply to cluster-to-cluster flows, facing more challenges though.

As a consequence, there needs to be a systematic study on flow relationships. First, it is necessary to have a better understanding on flow relationships. Questions such as to what extent flow relationships exist and what

types of relationships exist between flows need to be answered. Second, there is a need to establish a framework on how flow relationships can be exploited in general. Examination of potential performance improvements along with applicable situations needs to be done. Finally, investigation of specific techniques is essential to demonstrate how particular flow relationships can be exploited to improve applications. All of the above reasons constitute the motivation of this thesis.

1.4 The Thesis

Our central thesis is this:

> Internet applications continue to have performance issues even with the ever-increasing network bandwidth. Exploiting flow relationships and available packet space inside flows could help to improve application performance, but have not been well-studied. Our hypothesis is that classes of techniques can be deployed to exploit these relationships and enhance application performance with minimal costs.

We establish a framework on general approaches of exploiting flow relationships. The framework is based on two categorizations. In the first categorization, Internet applications are broken into different categories based on their traffic characteristics and performance concerns. We use a stage-based taxonomy to categorize commonly seen application stages into four types: bulk transfer, interactive, transactional, and streaming. An application session may include only one stage or can be composed of multiple

stages. By using this stage-based taxonomy, we look for general techniques that help a type of stage instead of a particular application. In the second categorization, we summarize the potential improvements that can be brought by exploiting flow relationships into four categories including reducing total packets, providing better information, avoiding timeouts, and reducing the number of RTTs. By combining the two categorizations together, the framework explicitly gives expected improvements and the applicable types of applications for a particular technique. This framework is important as it generalizes possible techniques that exploit flow relationships. Under this framework, we investigate several specific techniques and use them to illustrate how flow relationships are exploited in certain situations. Examples of these techniques are:

Piggybacking related domain names: we use this example to show how relationships between packets or flows are used to infer future traffic, and how this predicted traffic is piggybacked to ongoing traffic. Due to the internal content association and fixed application behavior, we find that much future traffic is predictable. In this example, we show high possibilities of related domain names being queried after an original domain name lookup. We propose to send answers to all related names in the response message for the first query, therefore avoiding future queries. This method not only improves latency performance for DNS, but also reduces the total number of query and response messages.

Using ACKs to send data: we use this example to show how available packet

space in flows can be used to improve transmission efficiency. As we have observed many ACK-only packets in current Internet traffic, there is potential to use them to send data. We find that using ACKs to piggyback data can achieve the same throughput as using two separate TCP connections or one traditional TCP connection which blindly sends data on both directions, while our method results in much fewer packets. This method is appropriate when a primary and secondary data direction exist. For example, a current P2P application "BitTorrent" has a "tit-for-tat" incentive mechanism that requires users to share their data resources at the same time data are retrieved from somewhere else.

Providing enhanced timestamp information: we use this example to show how available packet space can be used to provide better quality of information for applications. As TCP ACKs are sent back anyway, they can be used to provide better information to the sender. In this example, we propose an extended timestamp option, which provides more latency details than the current standard timestamp option. By using this extended information, the sender is able to distinguish jitter in one direction from the other as well as have a more accurate RTT estimation. We find that this method can help TCP to decide congestion conditions in each direction and to tune congestion windows more appropriately. It is especially useful for transmission schemes like TCP-Vegas [FCL01] and TCP-Westwood [WVSG02] that use delay jitter to estimate available path bandwidth.

Packet aggregation: we use this example to show another way to improve transmission efficiency. While the scheme of using ACKs to send data uses available packet space within one flow, packet aggregation exploits packet space across flows. Aggregating small packets together directly leads to improved transmission efficiency as well as reduced switching workload for intermediate routers.

Protecting critical packets: we use this example to show the potentials of using available packet space in related flows to avoid timeouts. Packet loss hurts application performance. For interactive applications like "telnet" or critical application stages like TCP connection establishment, a packet loss normally results in a long timeout. By using the available packet space provided by related flows, we could send duplicate copies or redundant data to protect packets that are sensitive to loss.

1.5 Outline of the Thesis

The organization of the thesis is shown in Figure 1.1. Chapter 1 is the introduction of the thesis. We discuss related work in Chapter 2 and talk about a background study of flow relationships in Chapter 3. Chapter 4 is the core of the thesis, in which we discuss a general framework of exploiting flow relationships. After that, we investigate five specific techniques of using flow relationships, each described in Chapters 5 to 9 respectively. At the end, we conclude our work in Chapter 10.

The outline of Chapter 2 to 10 is the following. In Chapter 2, we look

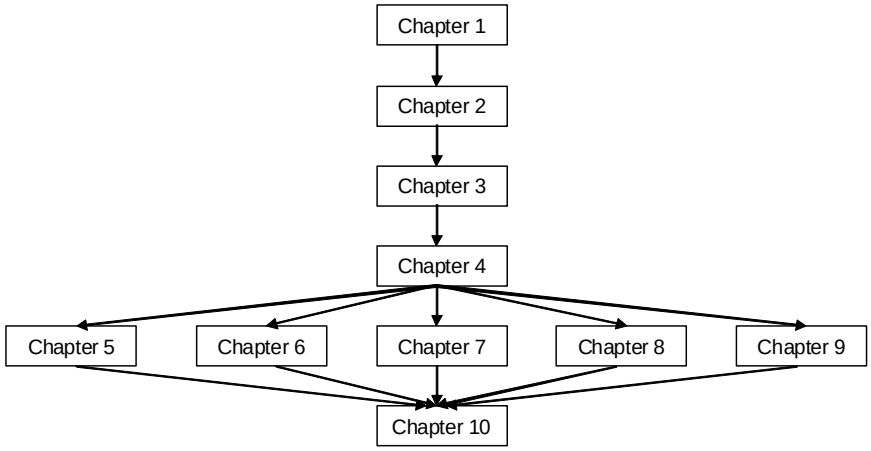


Figure 1.1: Organization of the Thesis

at the related work in five broad categories. We examine how previous techniques have been used in different layers, the types of relationships they exploit, and different scopes where they are applied. We also discuss the related taxonomy techniques for Internet applications.

In Chapter 3, we identify flow relationships by examining the temporal relation between flows. We investigate how many flow relationships exist between flows within the same host pairs or cluster pairs and how particular types of flows are related to others. In addition, we study the packet size characteristics inside flows and examine how much packet space is available.

In Chapter 4, we discuss possible performance improvements that exploit flow relationships and available packet space. We characterize these improvements and their expected benefits for applications. By using a taxonomy, we classify applications into different types and examine what

potential improvements fit best for each category. This chapter gives the framework of how flow relationships can be exploited.

In Chapter 5, we illustrate how a particular relationship between consecutive DNS queries is used to improve DNS performance. We discuss in details how the scheme works, implementation issues, and the performance gain.

In Chapter 6, we discuss a technique which piggybacks data with ACKs. We investigate how application performance and the resulting number of packets are influenced by this technique under a number of network paths.

In Chapter 7, we explore an approach which uses a bit of the available bandwidth to provide an enhanced timestamp option. We show how the enhanced information helps TCP to have a better understanding of current network conditions.

In Chapter 8, we look at a method of packet aggregation which intends to reduce the total number of packets on the Internet. We evaluate the gain of packet savings that can be brought by packet aggregation.

In Chapter 9, we examine the potentials to protect critical packet from loss by using available packet space to send duplicates. We investigate the possibilities that these duplicates can be piggybacked by concurrent flows under four example scenarios.

In Chapter 10, we discuss the future work that would complement our study and conclude the thesis with a summary of the work and its major contributions.

Chapter 2

Related Work

Even though flow relationships have not been studied in a systematic way, a number of previous techniques use flow relationships either explicitly or implicitly. In this chapter, we first look at four broad categories of how the techniques have been used in different layers, the types of relationships they exploit, and different scopes where they are applied. As we seek to apply the techniques of exploiting flow relationships in a general way, classification of Internet applications is also of our interest. In a separate category, we discuss related taxonomy techniques for Internet applications.

2.1 Shared or Centralized State Information

A network flow runs independently of other flows, which causes two problems. One is inter-flow competition where concurrent flows sharing the same bottleneck link compete for bandwidth when congestion occurs. Inter-flow competition hurts the overall throughput. Results in [CSA00, QZK99,

MB00] have shown that transmission time for the same amount of data over independently controlled TCP flows are much longer than that over TCP flows controlled in an aggregated manner when the network is congested. The other problem caused by the independence of network flows is that every TCP connection has to perform the slow-start procedure in order to gradually probe for available bandwidth, even though previous flows already have such knowledge. Much work seeks to avoid those problems by sharing state information among flows.

Work on Ensemble-TCP [EHT00] and shared TCP control blocks [Tou97] are ways for multiple TCP connections to share network information and better inform the TCP congestion control mechanism to avoid slow-start. An ensemble, no matter how many connections are part of it, is as aggressive as a single regular TCP connection in terms of getting network bandwidth. Inter-connection competition is avoided as all of them share the same congestion state. A newly established connection can directly use the current (if available) or the previous congestion window to avoid or speed up the slow-start procedure. This approach to congestion control has been implemented as part of the Linux kernel [SK02a]. This technique is good for concurrent TCP flows and is also useful for sequential flows if the shared information is retained. However, this approach is limited to traffic of one (the most prevalent) transport protocol.

Another approach to sharing is centralized scheduling of flows and packets. Work on the Congestion Manager (CM) [BRS99] is an example of this approach where a manager maintains congestion parameters and schedules data transmission for all flows ending with the same receiver.

The congestion control module inside the CM emulates the window-based congestion control scheme for a single TCP connection, whereas the scheduler module controls the sending rate of each flow by providing APIs for applications to adapt to network congestion. The centralized congestion control scheme eliminates the need for an individual application or protocol to conduct congestion control on its own, and at the same time avoids inter-flow competition. The CM mechanism does require the insertion of a new CM header between the IP and transport headers. This header is used to detect the CM capability on the receiver side and provide feedback about the transmission status to the sender.

By conducting coordinated congestion control over flows sharing the same bottleneck, the above approaches eliminate inter-flow competition as well as avoid unnecessary slow-start procedures. However, these schemes make little effort to aggregate flows and packets. Reducing the number of TCP connections helps alleviate the workload of servers as well as avoid performance overhead incurred by connection establishment and teardown. Reducing the number of packets helps alleviate the workload of the network, where less packets need to be routed or switched.

2.2 Aggregation and Multiplexing

Another class of work has looked at aggregating traffic at different levels. An approach to aggregate traffic at the application layer is to multiplex data streams on top of a TCP connection. HTTP/1.1 [FGM+99] is an object-wise multiplexing scheme, which uses a persistent TCP connection to fetch mul-

tiple objects. Another approach to this same problem is to bundle multiple objects in one response [WTM03, WMS01]. SCP [Spe] and SMUX [GN98] are two general-purpose session control protocols that multiplex data from applications on one TCP connection.

In essence, these application or session layer approaches use only one TCP connection. The inherent TCP congestion control mechanism is applied on all the multiplexed data streams. The TCP establishment and teardown as well as the slow-start procedure are only conducted once. However, these schemes introduce undesirable coupling, where logically independent data streams have to obey the syntax of a single TCP stream. The in-order delivery behavior imposed by TCP causes head-of-line blocking, where the loss of one packet prohibits release of successive packets, even though they belong to different data streams. Another problem of application level multiplexing is that it is not feasible for aggregating traffic from different applications and protocols.

The Stream Control Transmission Protocol (SCTP) [SXM+00] has multistream support, which allows multiple independent data streams to be multiplexed over one SCTP association. Unlike TCP, which enforces strict in-order delivery for the whole transmission, SCTP provides partially in-order delivery service where sequencing of messages is maintained on a per-stream basis. Message loss in one stream does not influence delivery of other streams. This mechanism resolves the head-of-line blocking problem caused by multiplexing streams over one TCP connection. At the same time, all streams within a single SCTP association are still subjected to a common congestion control mechanism. In addition, SCTP permits

bundling of more than one user message into a single SCTP packet, although SCTP can introduce a small delay as it tries to bundle. Users may disable bundling (like the PUSH flag is used in TCP) in order to avoid any delay.

SCTP is especially useful for applications that initiate multiple streams that have the property of independently sequenced delivery. An example is the delivery of a Web page which includes multiple in-line objects. On the other hand, a SCTP association may not be shared across applications, indicating that coordination among SCTP associations is still needed. For the same reason, bundling is only applicable for messages generated by the same application instance.

Another approach named "car pooling" [BS99] tries to aggregate packets at the network level. It suggests to place aggregation and splitting devices at desirable locations in the network. The aggregation devices merge small packets going to the same destination, while the splitting devices regenerate the packets at the destination. Packet car pooling reduces the total number of packets in the network, therefore alleviates the workload of intermediate routers whose processing cost is packet-based other than byte-based. By sharing the common packet headers, certain overhead can be avoided, resulting in improved transmission efficiency. The downside of the approach is the additional delay that is introduced in favor of packet aggregation.

Aggregation and multiplexing solutions at or above the transport layer avoid inter-flow competition as multiple data streams are now using one connection or association and subject to a single congestion control. This

kind of approach also minimizes the number of connections or associations, reducing the overhead incurred for each connection or association. However, multiplexing may introduce undesirable coupling and is only feasible for flows generated by the same application instance. Packet aggregation at the network layer seeks to aggregate packets instead of aggregating data flows. Its objective is to reduce the total number of packets other than to avoid inter-flow competition. As the network layer is below the application and transport layers, packet aggregation is not constrained to particular applications or transport protocols.

2.3 Coordination among Flows between the Same Pair of Clusters

We discussed approaches that perform aggregated congestion control or multiplexing for flows between the same pair of hosts (host-to-host flows). While similar approaches can also be applied to flows between the same pair of clusters (cluster-to-cluster flows), they present unique challenges. Because cluster-to-cluster flows may not terminate with the same hosts, TCP control block state parameters cannot be directly shared by connections ending with different hosts. For the same reason, APIs exposed by a central congestion control unit like CM [BRS99] are not applicable for applications residing on machines other than the one where the control unit exists. Multiplexing schemes like SMUX and SCTP can not even aggregate flows outside one application instance, let alone across different machines. Other approaches must be taken in order to achieve coordinated control

among cluster-to-cluster flows. We discuss this class of techniques in this section.

Most work for coordinating cluster-to-cluster flows introduces an aggregation point, which may be an individual box sitting in front of an edge router or the edge router itself. The aggregation point conducts aggregated congestion control over a collection of flows and regulates the sending rate of each flow. Based on whether the functionality is transparent to end applications or not, we further divide related approaches into two types.

The first approach type does not require participation of applications, where aggregated congestion control and traffic shaping are transparent to end applications. Both the Internet Traffic Manager (ITM) [MB00, DVM$^+$03, GM02b] and the Aggregated TCP (ATCP) architecture [PCN00] achieve such an objective by breaking the control loop between the two end applications into two control loops, one between the local application and the aggregation point and the other between the aggregation point and the remote application. The aggregation point allows state information sharing among all flows passing through it and regulates the flow sending rate by manipulating legitimate feedback (e.g. TCP ACKs) to sender applications. While ATCP is particularly designed for aggregated congestion control, the ITM architecture may carry other functions like flow isolation. Another approach of this type is the "TCP Trunk" scheme [KW99] where edge routers use management TCP connections to control the sending rate of data flows. The aggregated sending rate of all flows from a cluster is regulated by the sending windows of those management TCP connections. As this approach type is transparent to end applications, it does not complicate implemen-

tation of end systems. However, without the participation of applications, inter-flow tradeoff based on smart application decisions is not possible.

The other approach type allows applications to control the data sending rate for each flow by providing aggregated network state information. In Ott and Mayer-Patel's coordination mechanism [OMP02], they use an aggregation point (AP) on each cluster and insert a coordination protocol (CP) layer between the IP and Transport layers. An AP calculates network conditions based on all flows passing through it and conveys the information to end hosts by CP. Each flow decides its sending rate based on both current network conditions and pre-configured information. In [SAA+99], Savage et al. introduced an overlay network called "Detour" and each node in "Detour" can aggregate traffic from its local hosts over tunnels (TCP connections). Aggregate flow information can be shared by each individual flow. Approaches of this type seek to provide aggregate network information by a central node. The control of data transmission is left to applications, which obeys the ALF principle in [CT90]. On the other hand, implementation of the end system may be complicated. Current application implementations, as well as protocol stacks, may need to be changed.

Sharing congestion information across clusters may also be possible. In [Pad99], Padmanabhan outlines a receiver-based architecture in which a receiver sends congestion notification to senders that share a common bottleneck link. The senders may be scattered in different clusters. However, studies in this area are sparse and none of them has implemented a practical model.

2.4 Prediction

Approaches in the above classes are based on the observation that many flows occur concurrently or sequentially, therefore flows having the same WAN path can share state information between each other or be multiplexed over a single connection. However, these approaches did not look into the causes of these concurrent or sequential relationships. Concurrent or sequential flows may be caused by internal content relationships or the results of particular application behavior. These deterministic relationships can be used in prediction of future work. Observing one flow may be an indication of a future flow(s). Several approaches use these kinds of relationships to perform work in anticipation of future work.

As an example, the use of persistent connection specified in HTTP/1.1 [FGM+99] is a method to keep the current connection open in anticipation of future object transmissions. While this mechanism is embodied as object-level multiplexing (we have discussed it in Section 2.2), it is encouraged by the observation of the existence of internal relationships between in-line objects and the container page.

Previous work [WMS03] presents a methodology to use DNS queries to infer the relative popularity of any Internet server that can be identified by a name. It exploits the relationship that DNS lookups foreshadow the access of those Internet servers. The relative times a server name is looked up implies the relative popularity of the server. This approach seeks to explicitly use the relationship between a DNS flow and a successive data flow, although it does not try to improve performance.

Another approach that takes advantage of the relationship between a DNS flow and a successive application flow is the DNS-enabled Web (DEW) scheme proposed by Krishnamurthy et al. [KLR03]. In this approach, the unused portions of DNS messages are used to piggyback predicted HTTP requests and responses, therefore reducing the overall latency of the Web application.

Observation of flow patterns for a particular application can also be used as an indicator of whether the application performs optimally. Many short TCP flows only comprise one or two packets besides those used to open and close connections. TCP overhead is heavy for such a short transaction. T/TCP [Bra94] is one proposal to combine the TCP SYN and initial payload into one packet therefore saving the number of packets as well as avoiding one round-trip between sender and receiver. T/TCP was proposed for transactions, but currently only FreeBSD has implemented it and its usage is still in the experimental stage. Linux does not have plans for the implementation due to T/TCP's potential security problems [Han96]. T/TCP is more vulnerable to a sequence number attack and SYN flooding attack than a regular TCP which conducts a full 3-way handshake.

Prediction is commonly done at the application level, where content relationships and particular application behavior are known to infer relationships between flows and packets. By performing predicted future work, packets may be removed from the critical path of an application therefore achieving improved performance.

2.5 Related Taxonomy Techniques

An early study [CDJM91] looks at characterizing TCP/IP conversations. In this work, TCP applications are broken into two categories: bulk transfer and interactive. For example, FTP, SMTP, and NNTP are categorized into bulk transfer while telnet and rlogin belong to the other. This study gives the different traffic characteristics for the two sets of applications. It indicates that most packets for interactive applications are smaller than 10 bytes excluding TCP/IP headers and the packet inter-arrival time is mostly depended on user activities, i.e., key-stokes. On the other hand, packets from bulk transfer applications exhibit variety of packet sizes and their inter-arrival time is more dependent on transport protocols and physical characteristics of the network. While most of the observations still hold today for these particular applications, new applications that emerged after the early 1990's are not covered. For example, the Web, one of the most important Internet applications, does not seem to fit in either category.

Work like SURGE [BC98], RAMP [cLH03b], and Harpoon [SB04], focus on application characterization for traffic generation. Instead of analyzing application traffic at the packet level, they characterize source-level patterns in which data are sent. For example, Web traffic is modeled by SURGE as a sequence of requests for Web files with particular parameters following certain distributions. Similarly, RAMP models Web and FTP traffic with source-level descriptions of how applications generate this traffic. Finally, Harpoon uses a simplified model to simulate all Internet traffic, which is composed of many application sessions. Each session is represented by

a number of connections and each connection is associated with a file of a given size. Source-level modeling is another way to categorize applications. Applications sharing the same source-level model can be classified into the same category. The difference between source-level modeling and packet-level characterization is that the former is independent of network dynamics and underlying transmission schemes.

A more recent work [RSSD04] seeks to classify applications for different quality of service (QoS) treatment. It put applications into four categories: interactive, bulk data transfer, streaming, and transactional. This work distinguishes applications by their statistical signatures: packet size, session duration, and packet inter-arrival time. Applications that share the same set of signatures are grouped together. This classification method does not depend on port numbers to identify applications, but rather on the way in which an application is used. For example, the Web can be an interface for file downloading or interactive gaming. Simply using the application port can not distinguish these two cases.

2.6 Summary

In this chapter, we examined related techniques in five broad categories. The first four categories cover previous techniques of using flow relationships. The last category introduces related taxonomy techniques for Internet applications.

Observation of many concurrent and sequential network flows motivates much work to avoid inter-flow competition and provide aggregated

flow control. One class of approaches allow independent flows to share common state information. A specific central unit may be used to do aggregated congestion control and schedule flow transmissions. While these approaches successfully eliminate inter-flow competition and avoid unnecessary slow-start procedures of TCP, they do not reduce the total number of flows and packets.

Another class uses multiplexing or aggregation at different levels. Application or transport level multiplexing provides aggregated flow control as well as reducing the number of connections or associations. However, these approaches introduce undesirable coupling between logically independent data streams, which may cause head-of-line blocking. In addition, with most multiplexing schemes, it is not feasible to aggregate data streams beyond a single application instance. On the other hand, packet aggregation at the network level reduces the number of packets in the network, but does not try to avoid inter-flow competition. Network-level aggregation is not constrained to particular applications or transport protocols.

Approaches of the third class are designed for coordination between cluster-to-cluster flows. Among these approaches, one type requires participation of end applications while the other type is transparent to end applications. The former type grants better flexibility to applications for doing smart inter-flow tradeoff with the cost of more complicated implementations.

The fourth class of approaches exploit internal relationships between flows or packets to predict future events. These internal relationships may result from internal content relationships or particular application behav-

ior. By doing predicted future work, applications may have a shorter critical path, resulting in improved performance.

The last class of techniques are for classification of Internet applications. A taxonomy of Internet applications is important for generalizing techniques of using flow relationships. Taxonomy techniques can be quite different depending on what set of characteristics are used to classify applications. One group of taxonomy techniques look at the source-level patterns of how data are created, while another group focuses on packet-level characteristics observed in generated traffic.

Chapter 3

Background Study on Existence of Flow Relationships

As a background study for this thesis, we examine the existence of flow relationships in this chapter. We start with checking the temporal proximity between flows within several traffic logs. By breaking down flows into different types based on their corresponding transport protocols and ports, we further inspect relationships between particular types of flows. Thereafter, we check the number of *non-full packets* (packets that are less than the full Ethernet MTU size of 1500 bytes) inside each type of flow and examine the relationships only between flows that have a certain amount of non-full packets. Finally, we study a list of popular applications using packet traces in order to understand flow and packet behavior of particular applications. In the course of these studies, we also examine packet size distribution.

3.1 Existence of Flow Relationships

In this section, we examine the extent to which relationships exist among network flows between the same host pairs and the same cluster pairs. We have obtained two sets of trace logs. One set was collected at the link between the WPI edge router and a commercial ISP (henceforth called WPI logs). The tool argus [arg] was used to trace IP packets and combine these packets into flows based on common host, port and transport protocol. Each record in argus logs describes the number of packets and total bytes in each direction for a flow in the last minute. The other set was collected at the head-end of a cable broadband service provider who uses Motorola cable modem devices (henceforth called ISP logs). The tool tcpdump [tcpa] was used to trace all IP packets but only save packet headers. The details of these logs are shown in Table 3.1.

A *flow* is the collection of all packets over a period of time between the same source and destination pairs, using the same transport protocol (e.g. TCP, UDP) and associated port numbers. For a TCP flow, it is normally bounded by SYN and FIN/RST packets. For flows of other types including UDP, the end of a flow is decided when a period of idle time (we use five minutes in our experiment) is observed. To facilitate our study, we define a flow to include traffic in both directions. Flows influenced by boundary conditions due to the start and end of logs are not explicitly separated from other flows. However, only less than 1% of flows are involved in boundary conditions, which does not likely influence the tone of the results.

Among the five logs, four of them were collected at the end of year 2003

while the other one were gathered more recently in September of 2005. For the analysis we focused on TCP and UDP flows with 80%-90% of these flows for TCP traffic. Furthermore, we filter out flows that are incomplete. We define an *incomplete flow* as a flow that only has one-way traffic, or a TCP flow that has less than 3 packets in both directions. These incomplete flows are mostly caused by unsuccessful connection requests while some of them we suspect are sent by some automatic port scan programs like "nmap". The purpose of this flow relationship study is to find relationship patterns between legitimate flows generated by regular application sessions and find ways to improve the performance of these applications. For such a reason, we excluded these "incomplete" flows from the study. 27% to 70% of the flows are incomplete across these logs [1]. In Table 3.1, we exclude these incomplete flows and only show packet and flow counts for the rest of the flows.

Table 3.1: Log Description

Log Name	Time	Length (hour)	Flow Cnts	Flow/Min	Pkt Cnts	Pkt/Min
wpi1	Dec'03	24	10.1M	7.0K	687.3M	477K
wpi2	Nov'03	24	8.0M	5.5K	607.7M	422K
wpi3	Sept'05	24	11.7M	8.1K	696.1M	483K
isp1	Dec'03	9.5	5.7M	10.0K	245.6M	430K
isp2	Dec'03	3.5	2.1M	9.8K	82.8M	394K

A *flow relationship* is identified if there exists temporal proximity be-

[1] The percentages of incomplete flows are 38%, 27%, 37%, 70%, 53% for wpi1, wpi2, wpi3, isp1, and isp2 respectively.

tween two flows that share the same scope (same host pairs or cluster pairs). We use different interval thresholds to check the existence of a relationship. Let t denote the start time of a new flow and Δt denote a threshold value. We say a relationship exists if there is another flow at time $(t - \Delta t)$. In other words, using a threshold of zero catches concurrent relationships where a flow exists when a new flow begins. Using larger threshold values captures sequential relationships where a flow begins within the given time interval after a previous flow. Figure 3.1 shows an example of a concurrent relationship between flow1 and flow2, a sequential relationship between flow2 and flow3 with the threshold Δt, and no relationship between flow1 and flow3 with the same threshold.

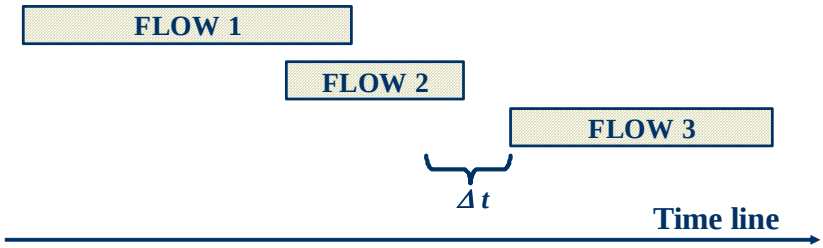


Figure 3.1: Illustration of Flow Relationships: flow2 and flow1 has a concurrent relationship while flow3 and flow2 has a sequential relationship given the threshold Δt

We examined the existence of relationships under different time thresholds as well as within different scopes for each of the five logs. Table 3.2 gives the results for the scope of host-to-host, where each cell is the percentage of flows that follow at least one previous flow between the same host pairs using different time thresholds. We see a significant relationship between flows even with thresholds as small as 10 seconds.

Table 3.2: Percentage of Host-to-Host Network Flows within a Specified Time Threshold of a Previous Flow

Time Period	Threshold			
	0 Sec	10 Sec	30 Sec	180 Sec
wpi1	39%	51%	56%	65%
wpi2	49%	59%	63%	71%
wpi3	28%	40%	43%	50%
isp1	27%	40%	44%	52%
isp2	27%	43%	47%	57%

Table 3.3 shows the results using the same data but under the scope of cluster-to-cluster. For WPI logs, we treat all WPI hosts as one cluster and group non-WPI hosts into clusters using BGP routes as others have done [KW00]. There are about 10% of IP addresses that do not have a corresponding BGP route. We cluster these IP addresses by applying traditional class C definition. For the ISP logs, all IP addresses are sanitized due to privacy reasons. The sanitization process scrambles each octet of an IP address separately. While we still have one-to-one IP address mapping, the network information of IP addresses is lost. For such a reason, we cluster IP addresses in ISP logs uniformly with traditional class C. In order to understand how the clustering method influences the results of flow relationships, we conducted experiments by using both clustering methods on WPI logs. The results show that 10% to 15% more flow relationships are found under the cluster-to-cluster scope when using BGP routes to cluster IP addresses than when using the traditional class C definition.

These results show that more than 40% of the flows exist in parallel with

Table 3.3: Percentage of Cluster-to-Cluster Network Flows within a Specified Time Threshold of a Previous Flow

Time Period	Threshold			
	0 Sec	10 Sec	30 Sec	180 Sec
wpi1	75%	83%	86%	92%
wpi2	77%	84%	87%	92%
wpi3	69%	77%	80%	87%
isp1	41%	51%	54%	61%
isp2	44%	54%	58%	65%

other flows from the same cluster and more than half of flows exist within 10 seconds of a previous flow from the same cluster. The two ISP logs show fewer relationships than those for the three WPI logs in the cluster-to-cluster scope. This observation is partially because the nature of the traffic is different between the two sets of logs. Another reason is that we use a more conservative clustering method for the ISP logs. When we also use traditional class C to cluster IP addresses in WPI logs, the percentages for WPI logs are reduced by 10 to 15%.

There are two special cases under the cluster-to-cluster scope. One case is that a local host initiates several flows to multiple hosts of a remote cluster. For example, a Web client retrieves multiple Web objects from the same site but different content servers. Another situation is that multiple hosts of a local cluster initiate flows to the same remote host. For example, local clients play Internet games using the same game server. We call the first case as *host-to-cluster* and the latter case as *cluster-to-host*. We distinguish these two special cases from other cluster-to-cluster relationships because

they both have a common end host. The common host has information for all related flows and therefore can work as a central control unit itself without introducing a separate cluster coordinator as done by [OMP02].

We show flow relationships for the host-to-cluster and cluster-to-host cases in Table 3.4 and 3.5 respectively. The results show considerably more relationships under the two situations than under the host-to-host scope. When combining the two cases together (adding the two cases together and subtracting the host-to-host relationships), we see most cluster-to-cluster relationships belong to either case. When comparing the two cases themselves, we do not observe an obvious pattern that one case is more dominant than the other.

Table 3.4: Percentage of Host-to-Cluster Network Flows within a Specified Time Threshold of a Previous Flow

Time	Threshold			
Period	0 Sec	10 Sec	30 Sec	180 Sec
wpi1	49%	61%	66%	74%
wpi2	57%	66%	71%	78%
wpi3	40%	51%	54%	61%
isp1	32%	45%	48%	56%
isp2	35%	48%	52%	61%

3.2 Relationships for Specific Applications

The results in Tables 3.2 and 3.3 show the existence of a significant number of relationships among network flows between hosts and clusters. We fur-

Table 3.5: Percentage of Cluster-to-Host Network Flows within a Specified Time Threshold of a Previous Flow

Time	Threshold			
Period	0 Sec	10 Sec	30 Sec	180 Sec
wpi1	49%	63%	69%	78%
wpi2	56%	67%	72%	80%
wpi3	37%	52%	58%	67%
isp1	34%	44%	48%	55%
isp2	33%	47%	51%	60%

ther broke down the network flows according to their related applications and studied the relationship between each type of flow. Compared with relationships shown for all flows, we found the relationships between different types of flows to be relatively stable and to exhibit a similar pattern for all the logs. Table 3.6 shows the results for a small sample of applications from the wpi1 log.

We identify the type of a flow by mapping the used common port number to its associated application as specified by [por]. This mapping method is accurate if a port number is used exclusively by one application and that application always uses the corresponding port. The first condition holds for most applications that have registered particular port numbers with IANA (Internet Assigned Numbers Authority). However, the second condition is more loosely followed. An application may use a port number other than the assigned one. Many current P2P applications do not use a fixed port number, instead they negotiate a dynamic port at the beginning of a session. When these cases happen, it is hard to catch all traffic

flows belonging to one application without the access to the application-level information of packets. In our experiments, we distinguish a flow for a particular application only if the flow uses the port assigned for that application. As a result, we may miss flows that actually belong to that application, but use other port numbers. However, the number of flows for an application does not change the tone of the results. For the five logs in Table 3.1, we see different flow counts for almost any application, but observe relatively stable relationships between different types of flows.

The first column in Table 3.6 is the application type based on transport protocol ("t" for TCP and "u" for UDP) and port number. Columns 2 and 3 show related flows that exist between two hosts for thresholds of 0 seconds (concurrent flows) and 30 seconds (a previous flow existed within the last 30 seconds). Similarly, columns 4 and 5 show related flows that exist between hosts in two clusters. We again show results for a threshold of 0 and 30 seconds.

The results shown in each cell of the table are the percentages of flows for the flow type in the first column that are related to other types of flows (including its own type). To conserve space we only list specific flow types when the relationship occurs for more than 10% of flows. In addition, we show cumulative percentages for all TCP and UDP flows.

The results in Table 3.6 show a number of relationships. A FTP (File Transfer Protocol) or SSH (Secure SHell) flow follows a previous flow of the same type in about 15% of cases within the same host pairs. The percentages become much larger under the scope of the same cluster pairs, where a FTP flow starts within 30 seconds of a previous FTP flow for over

Table 3.6: Relationship between Selected Flow Types for wpi1 Log

t: TCP; u: UDP; for example: u500 means flows using UDP port 500

AppPort	h2h:0s	h2h:30s	c2c:0s	c2c:30s
t21 (ftp)	tcp:3.3% udp:0.0%	t21:13.8% tcp:14.5% udp:0.0%	t21:63.1% tcp:64.7% udp:14.3%	t113:26.8% t21:93.7% tcp:95.0% u500:11.2% udp:25.5%
t22 (ssh)	tcp:9.1% udp:0.1%	t22:15.9% tcp:17.0% udp:0.1%	t22:24.7% tcp:44.0% u500:12.5% udp:18.5%	t110:18.1% t22:33.5% t443:14.6% t80:12.1% tcp:57.5% u500:15.1% udp:23.6%
t25 (smtp)	tcp:5.2% udp:0.0%	t25:18.0% tcp:18.1% udp:0.0%	t25:11.7% tcp:15.4% udp:9.4%	t25:30.0% tcp:34.9% u53:15.8% udp:16.9%
t80 (http)	t80:43.0% tcp:43.1% udp:0.1%	t80:58.6% tcp:58.7% udp:0.1%	t80:61.1% tcp:62.2% u53:10.0% udp:11.4%	t80:91.9% tcp:92.4% u53:14.1% udp:15.7%
t113 (auth)	t25:86.9% tcp:98.4% udp:0.0%	t25:86.9% tcp:98.5% udp:0.0%	t25:87.7% tcp:99.2% udp:5.3%	t113:11.2% t25:87.8% tcp:99.7% u53:13.7% udp:15.5%
t554 (rtsp)	t554:10.5% tcp:13.8% udp:4.5%	t554:53.1% tcp:55.7% u6970-7170:10.4% udp:11.8%	t554:47.3% t80:16.8% tcp:63.7% u6970-7170:38.8% udp:49.5%	t554:72.6% t80:23.0% tcp:82.2% u53:16.4% u6970-7170:42.6% udp:60.8%
t7070 (real-stream)	t554:40.0% t7070:36.0% t80:68.0% t8080:56.0% tcp:84.0% udp:0.0%	t554:76.0% t7070:72.0% t80:82.0% t8080:76.0% tcp:86.0% udp:0.0%	t554:40.0% t7070:36.0% t80:86.0% t8080:56.0% tcp:92.0% udp:2.0%	t554:76.0% t7070:74.0% t80:92.0% t8080:76.0% tcp:92.0% u53:10.0% udp:10.0%
u6970-7170 (real-stream)	t554:53.9% tcp:54.5% udp:6.7%	t554:54.0% tcp:54.6% u6970-7170:32.4% udp:34.2%	t554:54.0% tcp:54.7% u6970-7170:42.6% udp:45.2%	t554:54.1% tcp:54.9% u6970-7170:69.6% udp:74.5%
u27015-27017 (halflife)	tcp:0.0% udp:2.8%	tcp:0.0% udp:5.1%	tcp:1.5% u27015- 27017:11.0% udp:12.3%	tcp:2.2% u27015- 27017:30.9% udp:32.2%
u41170 (blubster)	tcp:0.0% udp:1.2%	tcp:0.0% u41170:13.0% udp:13.0%	tcp:5.1% udp:2.6%	tcp:7.3% u41170:15.5% udp:16.8%

90% of cases. Other types of flows like secure key exchange (using UDP port 500), Web (using TCP ports 80 and 443), are also often observed before a FTP or SSH flow within the same cluster pairs.

The authentication protocol using TCP port 113 is used 86.9% of the time with the SMTP protocol (t25). The DNS protocol (u53) precedes most applications when we consider cluster-to-cluster sequential relationships. HTTP flows (t80) are used 43% of the time concurrent with a previous HTTP flow between the same hosts and over 60% of the time concurrent with hosts in the same cluster. The last column shows that over 90% of HTTP flows occur within 30 seconds of a previous HTTP flow between hosts in the same cluster.

The "real" stream application normally uses multiple flows. The control flow using RTSP protocol (t554) is frequently seen in parallel with the data flows using either TCP (t7070) or UDP (u6970-7170). We also observe the TCP-based Real player streaming frequently occurs in temporal proximity to HTTP.

"Halflife", a popular on-line game, is often played concurrently or sequentially by different users within the same cluster. Network flows generated by "blubster", a peer-to-peer file sharing application, also show a fair amount of sequential relationships to its own type.

Again, we observe numerous flow relationships for particular applications and these relationships exhibit relatively stable patterns across all the traffic logs. These patterns are due to different reasons. For example, the relationship between DNS flows and other application flows is caused by application behavior. The relationship between one Web flow with another

Web flow is mostly because of the content relationship between Web objects. The observation of concurrent FTP flows or "half-life" game flows are due to user access activities. While the reasons behind these patterns may be different, they can all be used to infer future flows or packets.

3.3 Non-full Packets inside Flows

Exploiting flow relationships requires two conditions. The presence of flow relationships is one of them. The other requirement is the existence of available packet space in these related flows. If all packets are full, there is little use of flow relationships. On the other hand, if most packets are small, there is a big potential that the unused transmission capacity can help to improve performance and transmission efficiency of Internet applications.

We first study the packet sizes for both the downstream and upstream directions. We define *downstream* as the direction from an application server to an application client while *upstream* as the reverse direction. We observed that all WPI logs show a similar pattern while all ISP logs show another. Figure 3.2 and Figure 3.3 give packet size distributions for the wpi1 log and isp2 log respectively. We note that each record in a WPI log gives aggregated information for a flow in the past minute. It only provides the total packet count and byte count in that period instead of details for each packet. For this reason, we calculate the average packet size and use it to represent the size for all packets included in each record. If a record is the first or the last record for a TCP flow, we assume there is a SYN or FIN packet that has the minimal packet size and exclude the packet from

the average calculation.

Not surprisingly, most upstream packets are small. Around half of the upstream packets are pure ACKs. The average packet size for all upstream packets for both logs is around 400 bytes. For the downstream direction, the two logs show different patterns. For the isp2 log, over 60% of packets are either full (i.e. 1500 bytes) or empty (i.e. 40 bytes) and the average packet size is around 700 bytes. On the other hand, for the wpi1 log, most packet sizes are between 40 bytes and 1500 bytes with an average at 850 bytes. One reason for the difference between the two sets of results is that we use the mean packet size to represent all packet sizes of a record in WPI logs, which blurs the edges at both ends. Another reason is that we observed much P2P traffic in ISP logs while there is only a little in the WPI logs. Most packets involved in P2P traffic are either full for data chunks or empty for pure ACKs.

Compared with previous work [TMW97, MC00, FKMkc03, FML$^+$03], we do not see a clear tri-modal distribution in our study. This observation is largely because of the wide use of Ethernet with a 1500B MTU and that modern TCP/IP implementations use path MTU [MD90] instead of the default 576 bytes. A recent study [SPH05] also confirms that the mode of 576 bytes is not observed in traffic logs collected in several locations.

Despite the tri-modal distribution shapes, independent work shows compatible results of packet size characteristics. An earlier study [TMW97] on two MCI backbone links show only 10% of packets reach 1500 bytes while nearly half of packets are 40 to 44 bytes in length. A later longitudinal study [MC00] on traffic at the Ames Internet Exchange site also gives sim-

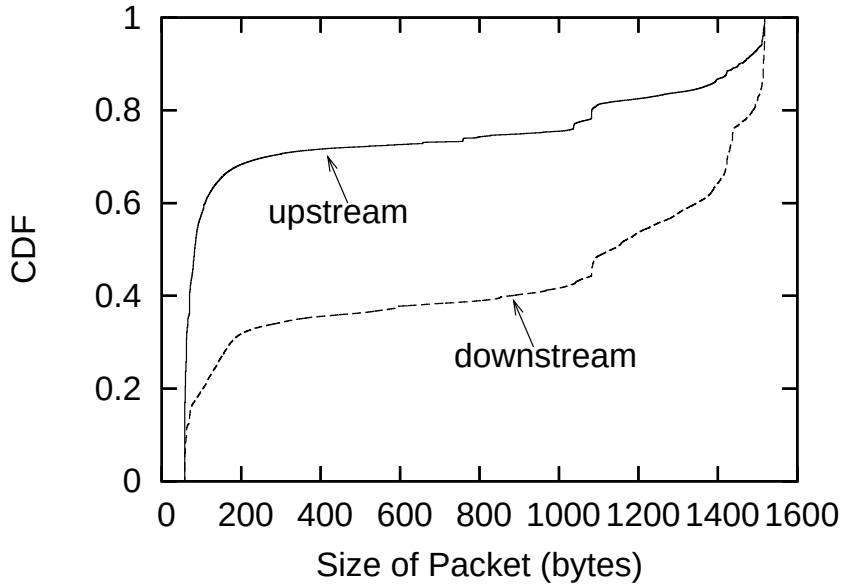


Figure 3.2: Packet Size Distributions for wpi1

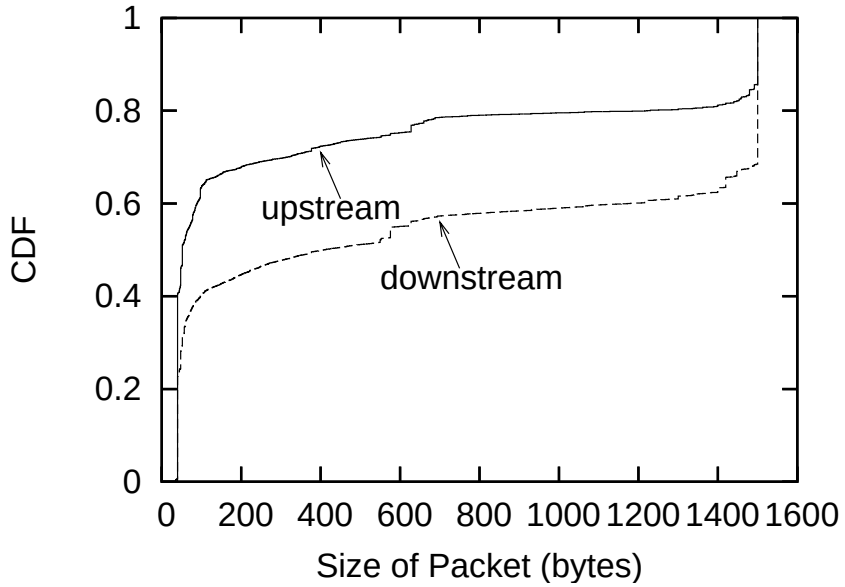


Figure 3.3: Packet Size Distributions for isp2

ilar statistics. Another longitudinal study [FKMkc03] based on NLANR PMA traffic archives indicates that the average packet sizes are between 500 to 1000 bytes for 20 sites. A more recent measurement [FML+03] from Sprint IP backbone shows various packet size distributions for a number of links. The percentages of full-size packets range from 10% to 30% while packets around 40 bytes are 10% to 70% for different links. A current ongoing study [SPH05] shows similar results for the two modes at 40 bytes and 1500 bytes. However, it also shows a strong mode at 1300 bytes for some links because of the usage of VPN (Virtual Private Network).

All of these results show a significant amount of unused packet capacity in current Internet traffic. Even for downstream traffic, there are still over 60% of packets that are less than the MTU size of 1500 bytes. The capacity will further increase once MTU sizes become larger. A proposal of jumbo frame [KSO+01] suggests using 9000 bytes as the MTU size for gigabit Ethernet in order to improve transmission performance. This MTU size is also being supported by more routers on their WAN interfaces. Jumbograms defined in IPv6 [BDH99] further extends IP payload size to exceed 64K bytes. There is great potential for this unused transmission capacity to be exploited for the purpose of improving application performance. Note that the use of VPN increases packet size due to the overhead incurred by IPSec (IP Security). In practice, packets to be sent over a VPN tunnel are recommended not to exceed 1300 bytes in order to avoid fragmentation once the IPSec header is added [SPH05]. However, even under the limit of 1300 bytes, there are still much available packet space. With the usage of jumbo frames, the much increased packet size overly counteracts the effect

introduced by the IPSec overhead.

While we have seen many flow relationships and much available packet space, a question is whether the available space can be combined with the flow relationships. In other words, only relationships between flows that have available space are useful. For this reason, we count a flow relationship only when both flows have over a certain level of non-full packets. We say a packet is a *non-full packet* if it is smaller than the Ethernet MTU of 1500 bytes. We define a *non-full flow* as a flow that has equal or above a certain amount of non-full packets. We use 20%, 50%, and 80% as thresholds to decide whether a flow has enough non-full packets. For example, the 50% threshold means that a flow relationship is counted only when a flow has equal or above 50% non-full packets and see another flow that also has at least 50% non-full packets. We show the results of non-full flow relationships in Table 3.7 for the wpi1 log under the time threshold of 10 seconds. Results for other logs and different time thresholds have the same tone and are not shown.

Table 3.7: Flow Relationships between Non-full Flows with Different Thresholds for the wpi1 Log under the Time Threshold of 10 Seconds

Non-full	Scope			
Pkt $\geq$	h2h	h2c	c2h	c2c
0%	51%	61%	63%	83%
20%	50%	60%	61%	82%
50%	46%	56%	58%	80%
80%	41%	52%	53%	77%

The numbers at the 0% line are copied from previous results when we

did not put the non-full packet constraint. It is used as the baseline for comparison with the results when such a constraint is applied. We do not see that flow relationships have changed much due to the non-full packet constraint. For the 20% threshold the change is trivial and for the 80% threshold the drop is less than 10 percent. A further investigation shows that most network flows satisfy the constraint of having enough non-full packets, which explains the minor influence of the constraint. We also observed that a small amount of flows have few non-full packets, but contribute significantly to the total packet count. Because these flows amount to only a small portion of the total flows, they do not influence the results much even after they are excluded from relationship check. However such flows do have a big influence on packet aggregation, which we will discuss in Chapter 8 .

3.4 Application Traffic Behavior

The results from the previous sections show that significant relationships do exist between network flows and most of these flows have a certain amount of non-full packets. However, the results do not reflect the specific traffic patterns of packets within a flow. Knowledge of specific traffic behavior is important as we look at exploiting the relationships between network flows. We studied the packet and flow behavior for a number of sample applications such as SSH, the Internet Explorer browser, RealAudio, RealVideo, Windows Media Player, QuickTime, network games, instant messaging, and electronic mail applications. From the results of this

study we make a number of observations about the behavior of applications:

- A single application often causes multiple flows to be created to the same host or hosts within the same cluster.
- Interactive applications such as SSH, games, and instant messaging generally use small packets. Applications over TCP use small packets in setting up a connection and sending acknowledgments.
- Many applications that use TCP have the PUSH flag set if the packet is less than full-MTU size, even if the packet may not need to be sent immediately to avoid control by Nagle's algorithm, which attempts to aggregate small amounts of TCP data [Nag84]. The setting of the PUSH flag causes the data to be immediately sent.
- The packet size for streaming applications depends on the encoding, but most packets we observed are not full. When TCP is used for streaming, the server always uses the PUSH flag.

While these observations are not novel, each of these is a factor as we examine exploiting the relationships between flows. In Chapter 4 we discuss general approaches of exploiting flow relationships and take these observations into account.

3.5 Summary

In this chapter, we have shown results from a background study on flow relationships. We found many flows between the same host or cluster pairs exhibit temporal proximity. This observation indicates a significant amount of relationships existing between network flows. We further quantified these relationships by examining the possibilities when two types of flows happen together. Results show that many types of flows are coupled due to application behavior or user access patterns.

The study on packet size characteristics shows that most packets are not full and many of them are small. The unused transmission capacity provides great potential in the sense of improving application performance. Much useful information can be sent without additional transmissions. After applying the constraint of having a minimal percentage of non-full packets, we still observe the existence of numerous flow relationships.

By monitoring packet interactions for several popular applications, we noticed several interesting facts. One is that many applications generate multiple flows, which directly leads to relationships between these flows. Another observation is that most packets are less than the full MTU size, which is partially because the "PUSH" flag is frequently used to defeat the delay caused by the Nagle's algorithm [Nag84].

Chapter 4

Framework of Exploiting Flow Relationships

The results in the previous chapter indicate the existence of numerous flow relationships across flows and much available packet space inside flows. These facts are encouraging as we seek to exploit flow relationships and enhance applications in terms of performance, quality, and transmission efficiency.

In this chapter, we establish a framework on possible techniques that exploit flow relationships and available packet space. By using a taxonomy, we break Internet applications into different categories based on their traffic characteristics and performance concerns. We then discuss the potential improvements that can be brought by using flow relationships for these application categories. Thereafter, we propose particular techniques that exploit relationships within a flow or across flows. Finally, we examine

where these techniques fit into the current TCP/IP protocol stacks.

4.1 Internet Application Taxonomy

In this thesis, we are looking to exploit flow relationships and non-full packet relationships in a systematic way. Instead of locking in a particular technique for a specific application, we search for general methods of using relationships. For that purpose, we need to classify both Internet applications and techniques that exploit flow relationships. We expect that a technique designed for one application is also applicable to other applications of the same type.

Applications can be classified quite differently depending on the goals set for the classification. In terms of using flow and non-full packet relationships, we seek to exploit the available packet space in or across flows. Consequently, we are interested in the following characteristics and categorize applications based on them.

- Packet-level statistics: what are the packet size characteristics? Small packet sizes indicate much available space in these non-full packets.
- Flow-level statistics: how many non-full packets are inside a flow? The number of non-full packets along with packet sizes give the total remaining capacity provided by a flow.
- Performance concerns: what is the most important factor that influences the performance of an application? This factor decides how we should exploit available packet space.

Having reviewed the previous taxonomy techniques in Section 2.5, we find none of them is fully appropriate for our objective. We see that the source-level modeling techniques such as SURGE, RAMP, and Harpoon, are good for traffic generation tools. However they include many application level details while lacking explicit information about packet level characteristics such as the packet size. Work of [CDJM91] splits TCP applications into either bulk transfer or interactive categories, where each category has different packet level characteristics. However, this taxonomy does not include UDP traffic, nor many current applications such as Web and streaming. The signature-based categorization method [RSSD04] gives insights of how application sessions differentiate from each other in packet size, duration, and packet inter-arrival time. However, it treats the whole duration of each application session uniformly and does not examine inter-flow relationships.

4.2 A Stage-based Taxonomy

Previous taxonomy techniques treat an application session uniformly and categorize applications based on the characteristics of the whole duration. The problem with these techniques is that an application session can have different characteristics in different time periods, each period having a different performance concern. For example, a FTP session, which is categorized as bulk-transfer in the previous work, is in fact composed of two sequential time periods. In the first period, transmission control and file meta information are exchanged between a client and a server. File trans-

Table 4.1: Different Types of Application Stages

Stages	Packet Size		Non-full Packets	
	Downstream	Upstream	Downstream	Upstream
Bulk Transfer	Big	Small	Few	Many
Interactive	Small	Small	Many	Many
Transactional	Small	Small	A Few	A Few
Streaming	Vary	Small	Vary	Many[a]

[a] If TCP is used for transmission

fer actually takes place in the second period. We observe different traffic characteristics during the two time periods. In the first period, the packets are sent back and forth in a "ping-pong" pattern and the packet sizes are generally small. User-perceived latency is sensitive to long round trip times (RTT) and packet loss. In the second period when bulk transfer takes place, traffic follows a packet-train pattern [CDJM91] and most packets are full. The tranmission performance is mainly determined by the available bandwidth and is more resilient to packet loss when fast retransmission and recovery schemes take control [Ste97].

In this work, we use *stages* to represent such time periods that have different traffic characteristics. In Table 4.1, we list four stage types that are commonly seen in application sessions, along with their characteristics. We follow the definition of "downstream" and "upstream" given in Section 3.3. Note that while we choose the same categories as those proposed in [RSSD04], we use them to distinguish application stages instead of separating application themselves.

The *bulk transfer* stage involves a large volume of data transfer over the

network without real-time constraints. Most packets in the downstream direction are full while the acknowledgment (ACK) packets in the upstream direction are mostly empty. The number of packets varies with the data size. For large data transfers, many full-size packets are generated in the downstream direction while a lot of ACKs are generated in the upstream direction. We often observe this type of stage in big file transfers using FTP, P2P, or HTTP. These applications are usually over TCP due to its reliable and in-order transmission features. Available bandwidth is the major contributor to the speed of bulk transfer, assuming that both TCP sending and receiving windows are big enough. While packet loss also influences the performance of this stage, in most cases it is just a reflection of network congestion where the sending rate is higher than available bandwidth. However, packet loss may happen independently of network congestion in certain situations such as over wireless segments with poor signals. The current TCP mechanism is not able to distinguish this situation and transmission speed may degrade significantly.

The *interactive stage* includes the traffic generated by multiple real-time interactions between a client and a server. The client behavior is generally controlled by human activities such as key-strokes. A telnet or SSH session, as an example, is composed of a single interactive stage. Packets in the upstream direction normally carry commands and packets in the downstream direction enclose responses. For both directions, packets are small and only a few packets are involved in each round trip. The duration of an interactive stage can be long and includes many back and forth packets. Sessions for many network games and Internet chatting (messenger) applications

are also composed of interactive stages. Due to the real-time constraint, this type of stage is sensitive to long round trip times. For applications over TCP, packet loss causes timeouts and retransmissions, which also add user perceived latency. Because of only a few packets in each round trip for most interactive stages, fast retransmission and recovery [Ste97] are not likely to take control.

The *transactional stage* comprises a small number of request-response pairs, which can be combined together to represent a transaction. Unlike the interactive stage, the transactional stage does not require human involvement and packets are normally generated by applications themselves. Examples like a domain name lookup or an email exchange session are composed of this type of stage. TCP connection establishment, as a special case, can also be seen as a transactional stage which includes three handshaking packets. Like the interactive stage, packets in the transactional stage are also small. However, unlike the interactive stage, the transactional stage is normally short and only includes a few packets. For the same reason as the interactive stage, the transactional stage is also sensitive to long round trip times and packet loss.

The *streaming stage* includes audio and/or video traffic with real-time constraints. Depending on the encoding methods of the multimedia data and the streaming applications, packet size in the streaming stage varies from half to full MTU size [LCK02]. The duration of a streaming stage also has a wide range from several minutes to tens of minutes [LCKN04]. For most streaming stages, a minimal available bandwidth is required in order to have acceptable performance. Streaming stages that only involve data

traffic in one direction such as audio broadcast or video on demand, long round trip time and delay jitter can be compensated by using a receiver-side buffer. However, for streaming stages requiring real-time interactions such as voice over IP (VoIP), long delay and delay jitter may cause severe performance problems. For situations when media data is ready before streaming, some applications push data as fast as possible at the beginning (called pre-buffering) and send chunks of data periodically afterwards [LCK02]. In that case, the pre-buffering stage can be seen as a bulk transfer stage and the rest of the session can be seen as a streaming stage.

These types of stages are abstracted from commonly seen applications. They may not be complete, nor be expected to cover all applications. For example, we do not study Internet game applications extensively. Particular game applications may not fit into any of the above categories. Also note that the same application may be composed of different stages depending on how it is used. For example, big file transfers using Web applications consist of bulk transfer stages while on-line transactions with Web interfaces are composed of interactive stages. In addition, the borders between these stage categories may not be strictly clear. For example, a VoIP session shares partial characteristics with both the streaming and interactive stages.

An application session can be composed of the four types of stages either sequentially or in parallel. The previous example of the FTP session can be seen as the concatenation of two consecutive stages: interactive and bulk transfer. A Web session, as another example, can be seen as an interleaving of transactional stages and bulk-transfer stages. In each transaction

stage, a connection is first established (if no previous connection exists or no persistent connection is in use) and then a request for an object is issued. The corresponding object is downloaded in the subsequent bulk-transfer stage. It is possible that an application session may comprise only a single stage. For example, a domain name lookup session can be seen as a single transactional session, which includes several request and response messages.

We need to note that stage and flow are two different concepts. We use stages to distinguish periods in an applications session that have different packet characteristics and performance concerns. On the other hand, we use flows to capture flow relationships within one application or across applications. A flow relationship may be induced by different reasons including application behavior, content relationships, and user access pattern. However, a stage only reflects the nature of an application. A flow may include multiple stages. For example, a ftp data flow includes a transactional stage for TCP connection establishment and a followed bulk transfer stage. We join these two concepts together by looking at the techniques that exploit flow relationships to help different types of stages.

4.3 Potential Performance Improvements

In the previous section, we defined four types of stages and use them to compose application sessions. In this section, we look for potential performance improvements achieved by exploiting flow relationships. As an application session may be composed of multiple stages and each type of

stage can have different performance concerns, we examine each potential improvement for its best applicable stages. In the later part of this chapter, we discuss a number of techniques that exploit flow relationships. We abstract the potential improvements that are brought by these techniques in several categorizes and list them in Table 4.2.

Table 4.2: Potential Performance Improvements

Potential Improvement	Applicable Stage(s)
Reducing total packets	All
Providing better information	All
Avoiding timeouts	Interactive Transactional
Reducing the number of RTTs	Transactional

Reducing total packets: we have observed many non-full packets in the packet size study conducted in Chapter 3. Even for a bulk transfer stage, much packet space is available from the ACK packets in the upstream direction. Could small packets be merged so that the total number of packets in the Internet is reduced? Reduction in the total number of packets brings several benefits. 1) The routing table lookup cost for each packet is irrelevant with respect to the size of the packet. Fewer packets means a reduced workload for immediate routers/switches. 2) Fewer yet bigger packets improve the buffering efficiency of routers. Most router implementations buffer packet headers and bodies separately [BS99]. Packet headers are stored in a faster memory while packet bodies are saved in a slower but cheaper

memory. The fast memory being expensive decides the packet queue size. 3) By letting packets for the same destination share the same IP header, a fraction of overhead can be saved.

Providing better information: our packet size study in Chapter 3, as well as other studies, show a significant amount of ACK-only packets for TCP flows. This phenomenon is mostly caused by the nature of TCP and due to that most applications only use a TCP connection to send data in one direction. As an ACK needs to be sent back anyways, can it provide better information than just an acknowledgment to previously received packets? A TCP connection could have a better understanding of network conditions and packet loss situations if provided with improved information. The TCP timestamp option [JBB92] and selective acknowledgment (SACK) option [MMFR96] are two examples in which higher quality of information than regular ACKs is provided. The former makes the RTT calculation easier and the latter gives more details of packet loss. There are other possible techniques in this direction. We will discuss them shortly.

Avoiding timeouts: for interactive and transactional stages, there is normally only one or two packets in each RTT. A packet loss will cause timeout and retransmission, which normally occurs long after a RTT. Avoiding long timeouts will help to reduce user-perceived latency. As packet sizes in both stages are small, we see two possibilities to avoid a long timeout. One is to let a timeout happen earlier than a normal timeout by piggybacking the retransmission with a subsequent

packet. This method may cause premature retransmissions, but not extra packets. The other method is to protect packets from loss by sending redundant data such as forward error correction (FEC). The protective data can also be piggybacked into other packets without introducing extra transmissions.

Reducing the number of RTTs: as we have seen potential improvements to avoid timeouts, is it possible to avoid RTTs as well? If we can predict future packets, we may send them ahead of time to avoid future RTTs. If the predicted packets can be piggybacked with the current ongoing traffic, the cost is minimal even if the prediction is wrong. It is hard to predict future packets for interactive stages which normally involve unpredictable human activities. It is also hard to conjecture prospective content for a streaming stage. For bulk transfer, even though prediction is possible, most packets in the downstream direction are full and piggybacking is not possible. As a result, the transactional stage is the only stage in which both prediction and piggybacking are achievable. As an example, the transactional TCP (T/TCP) [Bra94] proposes to send the request message and the FIN packet together with the initializing SYN packet. It exploits a simple relationship among the SYN, the request, and the FIN packets that normally exist in short transactions using TCP. In more complex cases, prediction needs the input of application-level information and flow relationships that exhibit a relatively fixed pattern.

We seek to achieve the above improvements by exploiting available packet space in existing traffic, therefore not introducing additional packets. However, these improvements do not necessarily reduce the total number of bytes. In general, transmission of a packet involves both per-packet and per-byte costs. The former includes the processing time for routing and switching in the intermediate routers. The latter includes buffering and transmission time incurred for each byte. For most techniques we will dicuss below, they do not introduce additional packets or even reduce the number of packets. However, these techniques do not seek to explicitely reduce the number of total bytes. In some cases, there could be more bytes introduced. We will discuss this issue in more detail as we examine each technique.

4.4 Techniques using Relationships within a Flow

After exploring potential performance improvements that can help applications in different stages, we now look at particular techniques that exploit flow relationships to achieve these expected improvements. There are two types of flow relationships. One type of relationships are between packets within a flow (henceforth called *intra-flow relationships*). The other type is between packets across flows (henceforth called *inter-flow relationships*). In this section, we focus on exploiting the first set of relationships. In Table 4.3, we list particular techniques that exploit intra-flow relationships under different categories of the potential performance improvements we have discussed in Section 4.3.

Table 4.3: Techniques using Intra-flow Relationships

Potential Improvement	Applicable Stage(s)	Intra-Flow Techniques
Reducing total packets	All	Data piggybacking [Chapter 6]
Providing better information	All	Enhanced ACK [Chapter 7]
Avoiding timeouts	Interactive and Transactional	Aggressive timeout
Reducing the number of RTTs	Transactional	Packet prediction [Chapter 5]

Data piggybacking: while TCP allows ACKs to be piggybacked in data packets, our technique uses ACKs to piggyback data. Previous studies [TMW97, MC00, FKMkc03, FML+03] and our packet size study in Chapter 3 show that nearly a half of TCP packets are only ACKs without any payload. As many ACK-only packets exist in the Internet, there is a good chance to piggyback a significant amount of data without introducing extra packets. This method fits best for the bulk transfer stage where many ACKs are present in the upstream direction. One scenario of using data piggybacking is in P2P applications such as "BitTorrent" (BT) [Coh03, Tur05]. The "tit-for-tat" incentive mechanism of BT requires users to share their data resource at the same time when retrieving data from somewhere else. We investigate this approach in greater depth in Chapter 6.

Enhanced ACKs: to help provide higher quality of information for applications, we suggest to have ACKs include better information than

what they currently do. Like data piggybacking, this method also exploits the available packet space in the ACK packets in the upstream direction. For example, the current TCP timestamp option [JBB92] provides an easy way to calculate RTT between two end hosts. However, we find that there is still room to extend this option to offer enhanced information. We propose to add timestamps to record when data packets are received in addition to the existing timestamps which record only when data packets are sent. By using this additional information, the sender can calculate one-way delay jitters as well as have a more accurate estimation of RTTs. We will discuss in more detail the enhanced timestamp option in Chapter 7. In general, enhanced ACKs help TCP to have a better understanding of network conditions, which is especially useful for applications that are sensitive to network dynamics, such as those that include bulk transfer or streaming stages.

Aggressive timeout: to avoid a long timeout when packet loss occurs, one way is to let the sender retransmit more aggressively. A drawback of aggressive timeout is that it may cause premature retransmissions. However, we may require that an aggressive retransmission only happens if it can be piggybacked into a succeeding packet, so that no extra packets are introduced. It is reasonable to assume that premature retransmissions do not occur often and retransmitted message sizes are small, in which case this method does not introduce many additional bytes. Aggressive timeout fits best in interactive and trans-

actional stages in which packet sizes are small and user-perceived latency is sensitive to packet loss. While we do not examine the aggressive timeout method within a flow, we do investigate the possibilities of sending duplicate packets using related flows in Chapter 9.

Packet prediction: it is possible to predict future packets by exploiting relationships between packets within a flow. These predicted packets can be sent with the current ongoing traffic if there is enough available packet space. If the prediction is correct, we save a number of RTTs. If the prediction is wrong, only minimum costs are incurred as no extra packets are introduced. The accuracy of prediction highly depends on the stability of relationships observed between packets. In Chapter 5, we illustrate how relationships between domain names are used to predict future queries with the help of application-level knowledge.

All the above four techniques exploit available packet space and packet relationships within a flow. Both the data piggybacking and enhanced-ACKs approaches use the available space provided in ACK packets in the upstream direction of a TCP flow. The former attempts to improve transmission efficiency while the latter seeks to provide better quality of information. Aggressive timeout tries to use temporal relationships between consecutive packets within a flow and utilizes a succeeding packet to piggyback an aggressive retransmission. Finally, packet prediction takes advantage of packet relationships that exhibit relatively stable patterns.

4.5 Techniques using Relationships across Flows

In the previous section, we have discussed approaches that use available packet space in a flow to help the flow itself. As we have observed many flow relationships across flows in Chapter 3, it is also possible to exploit the available packet space from all related flows. In this section, we examine how inter-flow relationships can be used to help with application stages. In the last column of Table 4.4, we list particular techniques that exploit inter-flow relationships under their corresponding categories of the potential performance improvements.

Table 4.4: Techniques using inter-flow Relationships

Potential Improvement	Applicable Stage(s)	Inter-flows
Reducing total packets	All	Packet Aggregation [Chapter 8]
Providing better information	All	Information Sharing
Avoiding timeouts	Interactive, Transactional, and Streaming	Critical Pkt Piggybacking [Chapter 9]
Reducing the number of RTTs	Transactional	Upcoming Flow Prediction

Packet aggregation: merging small packets is a way to improve transmission efficiency. For example, the Nagle's algorithm [Nag84] was designed for that purpose, but only for messages within one TCP flow. We can extend the scope to aggregate packets from multiple flows. For example, packets from a Web flow can be combined with packets

from another Web flow or a streaming flow as long as the merged packet size does not exceed the MTU. There are better opportunities to aggregate packets from multiple flows than from just one flow while still maintaining a reasonable delay in favor of aggregation. We examine the gain of packet savings by using packet aggregation in Chapter 8.

Information sharing: with the presence of flow relationships, it is possible to let a flow share information from previous or concurrent flows. Previous work such as Ensemble-TCP [EHT00] and Congestion Manager (CM) [BRS99] use a shared tcp control block (TCB) and a central unit respectively to facilitate information sharing among flows within the same host pair. By employing an aggregation point (AP) in each cluster, another study [OMP02] allows information sharing across flows within the same cluster pair. As an extension to the technique of enhanced ACKs within a flow, we may use ACKs or SYN packets to carry the information obtained from other flows. This extension allows information sharing among flows within the same host-to-cluster pair without demanding for an additional device such as an AP. A scenario for this type of usage is that in a Web session, in-line objects of one Web page are served by multiple servers in the same site. The first connection to the site has no previous information to use. But for subsequent connections, the same receiver can use SYN or ACK packets to provide to senders the information such as RTTs or congestion window sizes that are obtained from previous

connections. With the help of this information, a sender learns current network condition quickly and may avoid the TCP slow-start procedure.

Critical packet piggybacking: we have previously suggested to use aggressive retransmissions to avoid long timeouts and let retransmitted messages be piggybacked into successive packets in the same flow. With the presence of concurrent flows, there are also possibilities for these retransmitted messages to be piggybacked into packets from other flows. Aggressive timeout is especially helpful for interactive and transactional stages, which is sensitive to packet loss. More generally, an aggressive retransmission is used to protect a packet that is suspected of being lost during the first transmission. This packet is critical for performance as it has a high chance of being lost which will cause a long timeout. Critical packet piggybacking is a general technique that sends redundant data to protect packets critical to application performance. The redundant data are sent using the available packet space provided in the same flow or other concurrent flows, therefore no extra packets are introduced. As another example, the I frames in MPEG-1 video [MPFL96] are critical to streaming quality. The loss of one I frame causes all frames that depend on it useless. Due to the delay constraint, a retransmitted I frame may be too late to be useful. Schemes such as [WCK05, FB02, LC00] propose to protect these frames from loss by using the forwarding error correction (FEC). FEC codes, as another type of redundant data, may also

be piggybacked into other packets if space is available. We add the streaming stage into the applicable stages under the "avoid timeout" category in Table 4.4 as sending FEC is another way to avoid timeout. We discuss critical packet piggybacking in more details in Chapter 9.

Upcoming flow prediction: as we have observed that many flows have relatively fixed relationships in Chapter 3, it is possible to infer future flows from the occurrence of the existing flows. The previous proposed packet prediction approach uses packet relationships within a flow. This approach exploits relationships across flows. For example, the DNS-enabled Web (DEW) scheme [KLR03] uses the relationship between a DNS flow and a subsequent Web flow, therefore proceeding DNS messages may piggyback following Web requests and responses. As another example, in Web sessions we often observe many sequential connections to the same Web site. It is possible to predict future connections with the knowledge of site content. A connection establishment procedure can be seen as a transactional stage and it can be easily piggybacked by other transactional stages. In a simple way, we can use the SYN packet for the first connection request to indicate the intention to open multiple connections.

The above four techniques use relationships between flows and exploit available packet space presented in all concurrent flows. Both packet aggregation and critical packet piggybacking try to use temporal relationships between non-full packets from all concurrent flows. The former seeks to improve the transmission efficiency while the latter aims to improve the

transmission quality. Information sharing exploits relationships among sequential or concurrent flows and allows the information of one flow to be conveyed to others. Finally, upcoming flow prediction makes use of the relationships between flows that exhibit relatively fixed patterns.

4.6 Techniques by Levels

Previously, we have examined techniques that exploit intra-flow and inter-flow relationships. We organize them by their respective improvements. In this section, we discuss where are most appropriate places for these techniques to be implemented. In Figure 4.1, we put these techniques into three levels (layers): application, transport, and network.

<table>
<tr><td rowspan="3">Critical Packet Piggyback</td><td colspan="2">Packet Prediction</td><td colspan="2">Upcoming Flow Prediction</td><td>Application Level</td></tr>
<tr><td>Data Piggyback</td><td>Aggressive Timeout</td><td>Enhanced ACKs</td><td>Information Sharing</td><td>Transport Level</td></tr>
<tr><td colspan="4">Packet Aggregation</td><td>Network Level</td></tr>
</table>

Figure 4.1: Techniques by Levels

We put both prediction approaches at the application layer because application-level knowledge is normally needed for the prediction purpose. We have observed many consistent relationships between non-full packets and flows in Chapter 3. A significant amount of these relationships are caused by the relationships between content enclosed in these packets or flows.

We place the data piggybacking, aggressive timeout, enhanced ACKs,

and information sharing approaches at the transport level as both timeout and ACK are the concepts of this layer. Data piggybacking uses packet space in ACKs to send data. Enhanced ACKs and information sharing add more information to regular ACKs. Aggressive timeout avoids long timeouts by aggressively retransmitting packets that may get lost.

Packet aggregation fits best in the network layer. The network layer is the natural point to aggregate packets from different transport protocols and applications. It also facilitates the aggregation of traffic from different machines within a cluster, as the network layer itself also performs routing and forwarding functions.

Finally, critical packet piggybacking is across all the three layers. The application layer or transport layer decides which packets to protect and is also responsible for generating protective data. Once the data are passed down to the network layer, the network layer seeks opportunities to send them with other ongoing packets.

4.7 Summary

In this chapter, we established a framework on potential performance improvements that exploit flow relationships. By using a stage-based taxonomy, we categorize four type of stages that are commonly observable, including bulk transfer, interactive, transactional, and streaming. An application session may only include one stage or can be composed of multiple stages. By classifying application sessions into stages, we look for general techniques to help with a type of stage instead of a particular application.

Next, we examine potential performance improvements that help different stages in a number of ways, including reducing total packets, providing better information, avoiding timeout, and reducing the number of RTTs. There are two types of flow relationships to be exploited. One type includes the relationships within a flow. The other includes the relationships across flows. We examine four approaches that exploit intra-flow relationships as well as four other approaches that exploit inter-flow relationships. Among them, we will investigate the techniques of packet prediction, data piggybacking, enhanced ACKs, packet aggregation, and critical packet piggybacking in further details in Chapters 5 to 9 respectively. Finally, we discuss the best locations in the protocol stacks to place these techniques.

Chapter 5

Piggybacking Related Domain Names to Improve DNS Performance

In this chapter, we illustrate how a particular relationship between consecutive DNS queries can be used to improve DNS performance. This approach belongs to the category of *packet prediction* we discussed in Section 4.4. We use this example to show how relationships between packets or flows are used to infer future traffic, and how this predicted traffic is piggybacked onto ongoing traffic.

During our relationship study in Chapter 3, we found that a local domain name server (LDNS) [1] frequently sends more than one query to the same authoritative domain name server (ADNS) for different names within

[1] More strictly, a local domain name server should be called a recursive name server.

a short period of time. Using data from Chapter 3, we observed that over 40% of flows involving the DNS protocol for name server lookups result in multiple packet exchanges between client and server. At the same time, we also noticed that most DNS messages are smaller than the allowed size of the UDP packet in which they are carried.

As a means to reduce multiple-packet DNS flows between a local DNS server and an authoritative DNS server, we hypothesize that in many cases the authoritative DNS server can predict subsequent requests by a local DNS server based on knowledge of site usage and history of its DNS accesses. For example, the content of Web pages at busy Web sites is often served by multiple servers at the site, each with distinct names. Similarly, streaming or Instant Messaging applications use their own servers and are often combined with access to Web servers.

If the ADNS can piggyback resolutions of those related names in the response to the first query, it will save the LDNS from sending further queries. We call this approach *Piggybacking Related Names* (PRN). It benefits end-users as they experience less latency for DNS lookups. It also benefits ADNSs as they receive fewer DNS requests. Assuming that the piggybacked resolutions do not require additional packets then the approach reduces the number of packets needing to be routed through the Internet resulting in less congestion.

5.1 Background

DNS is a distributed database providing mappings between addresses and names [Moc87a, Moc87b]. The domain name space is a hierarchical structure with a group of root name servers at the top of the hierarchy. Below the root servers are generic Top Level Domains (gTLD) servers, which are delegated servers for domains such as ".com" and ".edu". The gTLD servers in turn can delegate their sub-domains to other name servers and so on. These domains are also called zones and these delegated servers are called authoritative servers for their assigned zones. Each zone can have a set of resource records (RRs) associated with it. There are many types of RRs. The most common type at an ADNS is an "A" RR that gives the mapping from a domain name to an IP address. For each RR, there is an associated time-to-live (TTL) parameter that indicates how long the RR can be cached.

The most common function of DNS is to return IP addresses for a given domain name. The process is typically initiated by a local application that calls an underlying resolver routine and passes the name as a parameter. The resolver sends a query to the local domain name server (LDNS), which in turn sends queries to responsible domain name servers if RRs for the name are not cached locally. As needed, the LDNS iteratively communicates with a root server then to a gTLD server then the authoritative domain name server for the name. Once the LDNS obtains the resolution, it returns the result to the caller application.

A domain name lookup is required for any application that identifies servers by names instead of IP addresses. The latency incurred by the

lookup procedure can influence the application's performance as a whole. In a previous study [WS00] we found that the median and mean lookup time for non-cached domain names is on the order of several hundred milliseconds. About 20% of the lookups took more than one second. Cohen et al. [CK02] indicates that DNS lookup time exceeds three seconds for over 10% of Web servers. This result is consistent with the measurement conducted by Chandranmenon and Varghese [CV01]. Jung et al. found that 10-20% of DNS lookups take more than one second based on traces collected in MIT and KAIST [JSBM02]. A more recent study shows the average latency for resolving non-cached domain names ranges from 0.95 seconds to 2.31 seconds for a variety of clients [LSZ02]. All of these measurements suggest that the DNS lookup time for non-cached domain names can influence the performance of interactive applications.

Caching is effective for reducing user perceived latency and is commonly adopted by DNS implementations. Previous studies [WS00, JSBM02, JC03, CK01, CV01] have shown cache hit rate varies from 50-90%. However with the increasing number of networked applications, there are many DNS requests generated for a single application and many of the cached copies of DNS mappings have a short time-to-live value in the cache. The result is that many requests for non-cached or stale DNS entries still exist. For example, in one day of WPI network flow data we observed over 40,000 DNS flows per hour. Thus further reduction of cache misses is still necessary for improving application performance.

5.2 DNS Latency

A central question about any proposal to improve DNS performance is to understand to what extent DNS performance is an issue. Recent work on DNS performance found that the average time to resolve non-cached domain names ranged from 0.95 seconds to 2.31 seconds for a variety of clients [LSZ02]. Previous studies had also found 10-20% of DNS resolution times greater than one second [WS00, CK02, CV01, JSBM02]. These collective results indicate that DNS latency performance is an issue for applications.

In conjunction with our primary work on improving DNS performance, we performed a study in June 2004 and again in February 2005 to better understand current DNS performance for a subset of locations. We used 20 LDNSs that we identified as part of previous work [WMS03]. These servers are all located in the United States and comprise four categories: commercial sites, educational sites, Internet Server Providers (ISPs) serving commercial companies, and ISPs serving home customers. Servers in the first three categories were found by using the `dig` tool to obtain the ADNS for an institution and then using directed DNS "A" record queries to determine if these authoritative name servers also played the role of LDNS for the institution. Servers in the last category were found by using published addresses for DNS servers of ISPs known to serve home customers.

For testing DNS performance, we used a list of just over 5000 unique names randomly drawn from 164,000 domain names from the logs discussed in Section 5.5. This list initially contained only valid names that were successfully resolved by using a local client, although in the second

study about 100 of the names were no longer valid. Each test involved using the `dig` DNS tool to direct a DNS query for each domain name to each of the 20 LDNSs. There are two possible situations. First, a LDNS could have the resolution for a domain name cached, in which case the total time is simply the round-trip time (RTT) between the `dig` client and the LDNS. Second, the LDNS needs to recursively perform a DNS lookup before returning the resolution. As part of the study we repeatedly measured the time for an entry known to be cached at a LDNS for establishing a baseline RTT measure between our `dig` client and each LDNS. We subtracted the mean of these RTT measures from all resolution times to determine the DNS resolution time by the LDNS for a given name. This is a similar approach as used in [GSG02] of using DNS to measure the RTT between arbitrary points in the Internet, but we use valid domain names rather than randomly generated names.

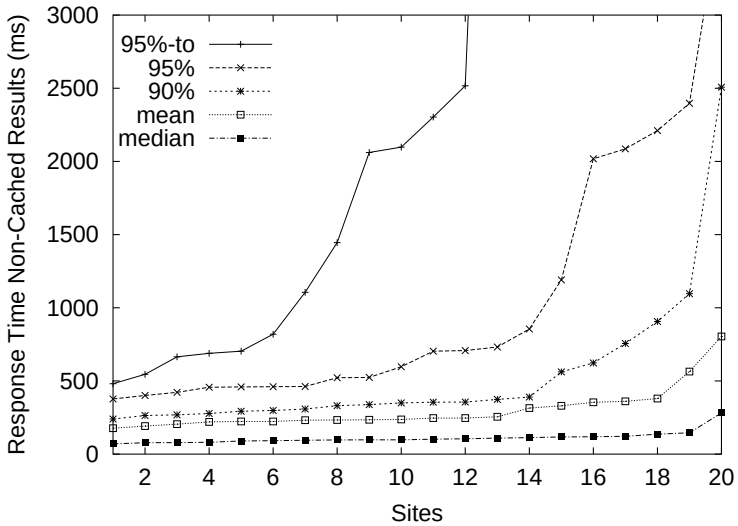
To distinguish between cached and uncached entries during analysis of the results, we first examined the distribution of resolution times from our separate study of entries known to be cached. For most LDNSs, the difference between the minimum and 95% RTT value was within 10ms and we used the 95% RTT value to determine cached/uncached entries. For LDNSs with more variation in RTT values, we used the mean RTT value as the threshold. We confirmed the validity of these time-based thresholds by comparing results against one where we checked for the presence of the authoritative Time-to-Live (ATTL) value in the returned entry to determine non-cached entries. We had independently found the ATTL value for each entry. We were able to make this check for the latter timeframe in our study.

We applied these time-based thresholds for cached/uncached entries for both timeframes in the study.

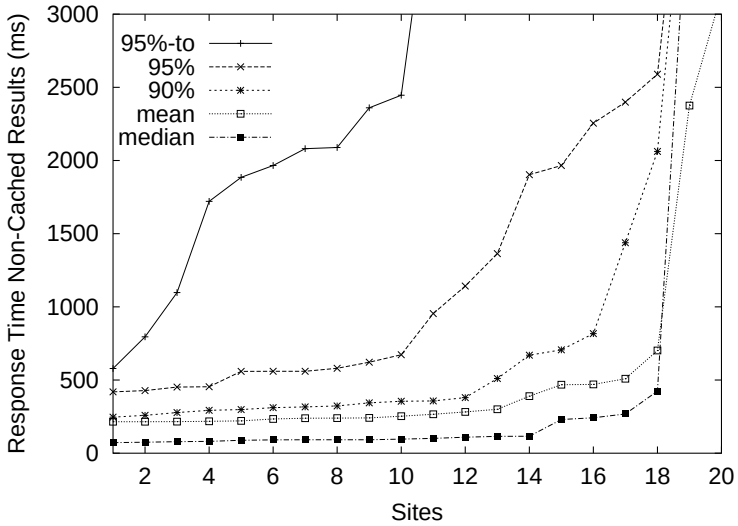
We used results for completed requests only with mean, median, 90% and 95% values for the 20 LDNSs in rank order shown in Figure 5.1. More than 5% of the queries did time out for six of the 20 LDNSs. We determined an effective timeout rate for each LDNS by subtracting the timeout rate to the LDNS itself from the overall timeout rate. The "95%-to" values in Figure 5.1 show the rank order 95% levels if the effective timeout rates are included for each LDNS.

An observation about the results is that the average response time is generally between 200 and 300ms for both timeframes, which is consistent with [WS00], but less than the average times on the order of a second reported in [LSZ02]. This difference may occur because the tested LDNSs are all in the relatively well-connected United States as well as the vast majority of the server names, so better average results than [LSZ02] are not surprising. The results do show that over half of the tested LDNSs exhibit a 95th percentile response time of over a half-second and 25% of the LDNSs yield a 95% response time over one second. If we include the timed out responses for valid server names, then the majority of the LDNSs have a 95% response time of greater than one second with about half over three seconds.

The outcome of this study is that the current average DNS latency is generally in the range of 200-300ms, but poor DNS performance is still a problem. While we do not know the source of all latency problems, potential causes are packet RTT variance and loss combined with relatively



a. June 2004



b. February 2005

Figure 5.1: DNS Response Time for Non-Cached Results

long timeouts typically used by DNS clients. In addition, [PPPW04] found that request overload and competition from periodic tasks can cause response problems for DNS servers. Approaches that reduce the amount of DNS traffic will improve the overall response time for applications. A clear direction for future work is to consider extending this methodology to a wider range of LDNSs with more work on the methodology to better understand how to handle the effects of caching and timeouts.

5.3 The Piggybacking Related Names Approach

As a means to improve DNS performance, the *Piggybacking Related Names* (PRN) approach is motivated by the observation that many applications and related applications generate a sequence of DNS requests for "A" resource records to the same ADNS for domain names within the same DNS zone of authority. The approach exploits the observation that most DNS packets are smaller than the allowed size of the UDP packet in which they are carried and hence there is potential for ADNSs to include "Additional Records" in response to a client's request. In RFC1034, it says the "Additional Records" response field "carries RRs which may be helpful in using the RRs in the other sections." In our work we propose that this field can also be used to contain additional RRs that the ADNS expects the client to subsequently request based upon the current request. As long as including the additional records does not exceed the maximum allowed size of a DNS packet then these additional records are delivered to the client with no additional packets and minimal cost on the packet-switched Internet.

Figure 5.2 illustrates the approach with the query/response dialogue between a LDNS client and an ADNS for resolution of multiple server names. In this example, names from a1.b.c to a5.b.c belong to zone b.c. The first query (in this example is a1.b.c) triggers a response that includes resolutions for the additional names in the zone. Once it obtains the response, the LDNS caches all included entries. Queries for names a2.b.c, a3.b.c, a2.b.c and a4.b.c will be cache hits. We assume the interval between T6 and T1 is bigger than the authoritative TTL (ATTL) for a2.b.c. So at T6, the entry for a2.b.c is stale and a query for it causes a cache miss. A cache miss causes a new query to be sent to the ADNS by the LDNS.

It is obviously not practical for an ADNS to piggyback resolutions for all the names in a zone. In reality, an ADNS only needs to piggyback resolutions it expects to be used in the interval before the next query is needed. In the example both "T1 to T6" and "T6 to T8" are such intervals. In Section 5.6 we study the expected number of these additional records as well as the amount of available room in a DNS response.

A clear advantage of the *Piggybacking Related Names* (PRN) approach is that it requires no changes to the existing DNS protocol while reducing the amount of DNS traffic for local and authoritative DNS servers. However the approach does require changes to the implementation of LDNSs and ADNSs. An ADNS must determine which names to be piggybacked and add them to the Additional Records section of a response message. This determination can be based on existing DNS queries as well as from knowledge of the site contents. A LDNS must extract the additional records and store them in its cache, which could be a problem in unnecessarily filling

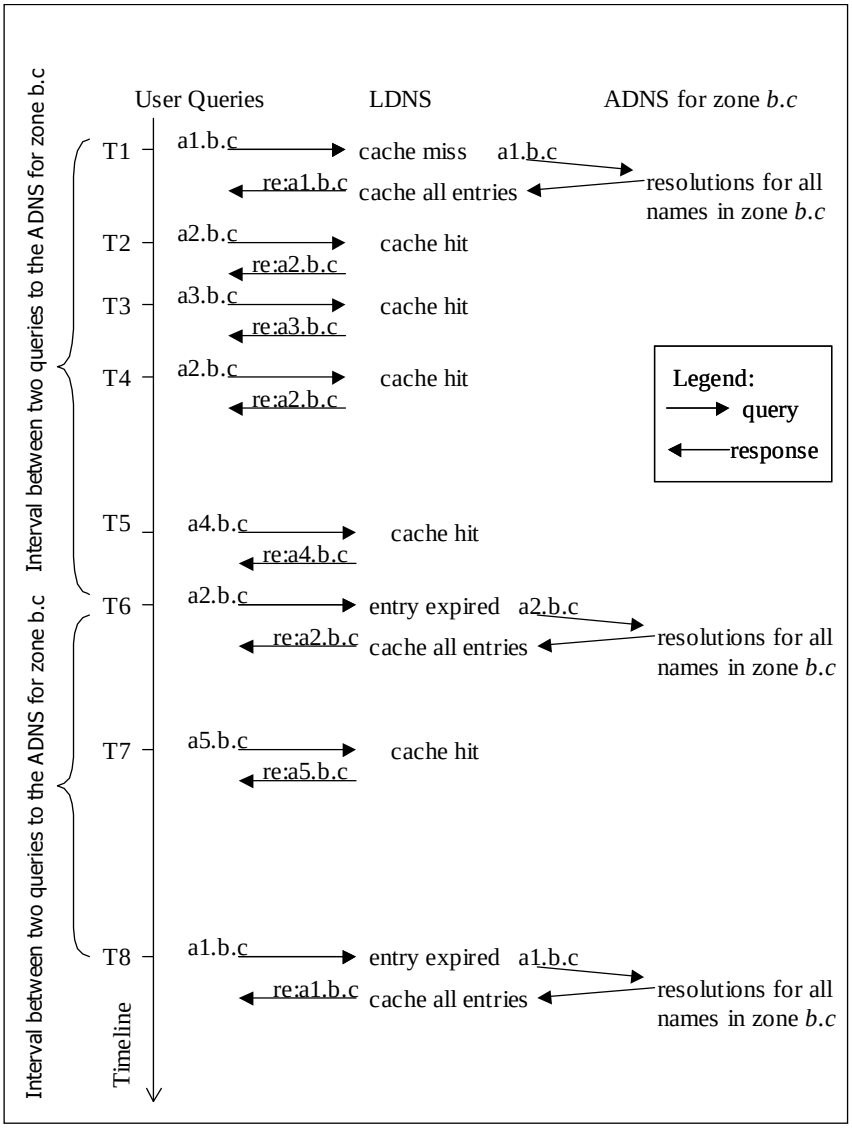


Figure 5.2: Illustration of the Piggybacking Mechanism

up the DNS cache, but in practice DNS cache records are small and cache space is not expected as a limitation. In our experiments, we assume all piggybacked records can be cached and will not be evicted before they expire. In the situation when cache space is limited, those piggybacked records can be tagged and be the first to be replaced if the cache is full.

One potential security issue with including resource records in the additional records field is a DNS-based attack called "cache poisoning" that is caused by allowing non-authoritative RRs to be cached by LDNSs [CER]. Our approach does not lead to this problem because a ADNS only piggybacks RRs for which it is the authoritative server.

5.4 Related Approaches

Previous research has examined other approaches for reducing the cache miss rate at a LDNS. This section discusses three proposed approaches and compares them with the PRN approach.

One approach to improve DNS performance is for clients to pre-resolve server names [CK02]. This approach requires applications such as Web browsers to predict, based on Web content, which DNS lookups will be required and to issue those lookups before the content is retrieved. While this type of predictive policy is similar to the server-side predictions of our PRN approach it requires changes in applications and allows predictions to be made based only on client-available information.

A second approach is to use separate DNS queries to renew stale DNS cache entries [CK01]. This approach has the advantage that these queries

are done outside of the critical path of an application and will improve the performance of an application. The problem with this approach is that it can generate many DNS queries for which the result is never used.

The third approach is to piggyback requests for stale entries onto a needed request to an ADNS [JC03]. This "renewal using piggybacking" (RUP) approach causes no additional DNS packets to be generated, but requires each LDNS to organize all resource records according to their zone. As in previous methods it also causes resolutions of names that may not be used again. This approach also requires that the DNS protocol support more than one request in a message.

To compare these approaches we examine the types of cache misses that they avoid. Using the terminology of [CK01], cache misses can be divided into two types: "first-seen" (FS) misses, indicating the first lookup of a DNS name; and "previously-seen" (PS) misses, indicating entries that have been previously seen, but expired. The two renewal approaches only reduce PS misses because they can only renew entries that have been seen before. The pre-resolving approach of [CK02] can reduce both FS and PS misses, but will only do so based on the immediate needs of the application. The PRN approach not only reduces FS misses based on server knowledge, but if those entries are already cached, it can be used to restore these entries to their full TTL duration, thus reducing PS misses.

5.5 Potential Impact

Before looking at the details of implementing the PRN approach, an important question is to examine its potential impact in terms of miss rates. It is known that LDNS caches satisfy over 50% of DNS requests received from local applications. With these hit rates, an argument can be made that DNS performance is not a problem. However, a number of factors justify the need to further reduce the number of non-cached lookups. First, despite the high hit rates, a substantial number of DNS queries must still be satisfied by contacting the appropriate ADNS and as previously mentioned 40% of the DNS traffic in WPI flow data indicate multiple DNS requests. Second, our own study in Section 5.2 along with recent studies have shown that latency is an issue for a portion of requests. Third, in the presence of a dropped packet the delay is much larger as LDNSs use a three or five second timeout. Fourth, more applications leads to more domain names at a site that must be looked up and many of these names carry shorter ATTLs to allow flexible load balancing.

We used three logs summarized in Table 5.1 to study the performance of DNS and examine the potential impact of improvements. The first log is of data from WPI's primary DNS server, which serves as both a LDNS and ADNS for the campus. For our purposes the log was filtered to only consider queries from WPI clients that are handled by the server in its role as a LDNS. We augmented these data by fetching the ATTL and the ADNS(s) for each unique name in the log.

The other two logs are generated from two NLANR Web traces [NLA]

Table 5.1: Summary of Trace Logs Used

Name	Queries	Date	Dur.	From
WPI	1169569	Apr '03	28 hrs	WPI DNS
RTP	1041275	Oct '03	7 days	NLANR
SJ	457070	Oct '03	7 days	NLANR

as done in [CK01]. Each entry of the Web trace is a request to an object identified by a uniform resource locator (URL). We extract the host name part from a URL as a query in a DNS trace. Because many browsers themselves cache name-to-IP address mappings for a short time, we make the same assumption as in [CK01] that there is no DNS lookup incurred if the same name is requested again within a 60-second window.

We used these three logs along with the augmented data to determine the miss rate performance of DNS using a trace-driven simulation assuming the cache is empty at the beginning of the simulation. The simulation mimics the regular behavior of a DNS cache as well as an ideal behavior where whenever an ADNS receives a query, it returns resolutions for all the names in its zone. Subsequent queries that belong to this zone are satisfied locally as long as these entries are still fresh.

Results in Figure 5.3 show that 26% of requests in the WPI DNS log result in misses and this percentage can be potentially reduced to 10% for a relative improvement of over 60%. Similar results are shown in Figure 5.4 for the RTP log where the percentage of total misses is over 45% with a potential reduction to under 25% for a relative improvement of about 50%. Similar results were obtained for the SJ log and are not shown.

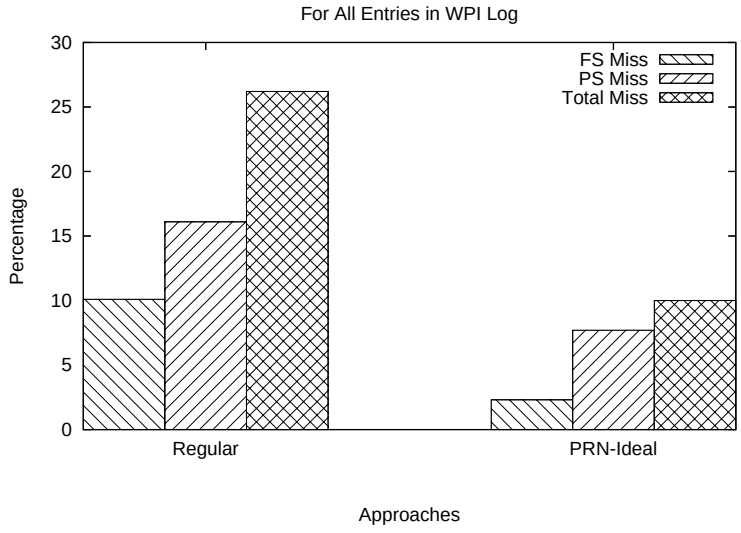


Figure 5.3: Potential Performance Improvement for Ideal PRN Policy with WPI Log

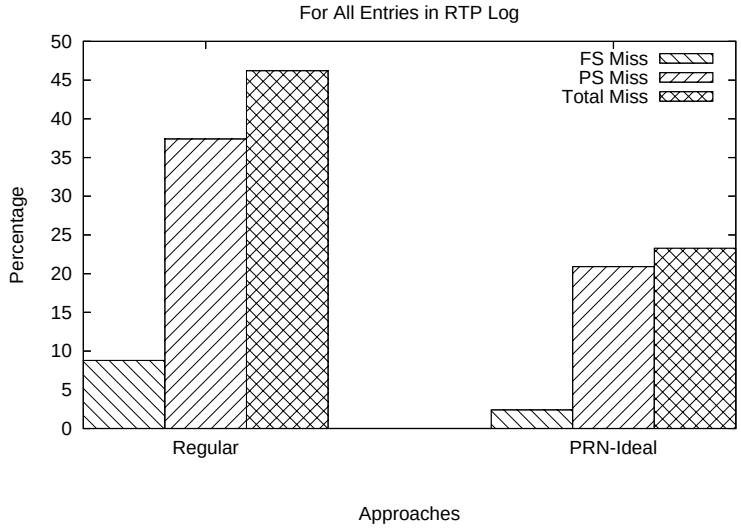


Figure 5.4: Potential Performance Improvement for Ideal PRN Policy with RTP Log

The collective results show significant reductions are possible in reducing both first-seen and previously-seen misses. Prediction of first-seen requests are not possible in renewal-based approaches while prediction of previously-seen requests extend the lifetime of the corresponding cached entries.

Note that adding related names does increase the packet size of DNS packets. However, because DNS type A RRs are generally small and only a few related RRs needs to be added, we do not expect that the inclusion of these related RRs will introduce many additional bytes. Given the potential of 60% reduced cache miss rate and the same percentage of reduced packets, the cost in bytes is well compensated by the saving of improved performance and the reduced number of packets. In the following, we discuss how many RRs can be piggybacked with the current size constraint of DNS packets and policies to pick related RRs.

5.6 Implementation and Policy Issues

Having established the potential usefulness of the PRN approach, in this section we discuss specific implementation issues regarding the number of resource records that need to and can be piggybacked on a DNS response. We also describe specific policies for an ADNS to make decisions on what records to piggyback and what information the ADNS must maintain for these policies.

5.6.1 Piggybacked Responses

We used the data from the WPI DNS log to determine the number of responses that would ideally be piggybacked on a response. We used the request intervals in Figure 5.2 to define DNS "bundles" for a zone. A DNS bundle includes all unique server names for the zone that occur in a request interval. The size of this bundle determines the number of DNS responses that would be useful for the ADNS of the zone to return.

Using this definition, we found about 20,000 DNS bundles in the DNS log. Figure 5.5 shows the cumulative distribution function (CDF) for the number of names inside each bundle. As shown, about half of the bundles have only one name—the response itself—while the other half have two or more names. The results show that only 5% of bundles have more than 15 entries and only 15% of the bundles have more than 5 entries. These results are encouraging for the PRN approach as they indicate it is useful for half of the bundles and the number of names that need to be piggybacked is not large.

5.6.2 DNS Response Message Capacity

DNS messages are limited to 512 bytes in size when sent over UDP [Moc87b], however DNS extension mechanisms [Vix99] extend the limit to 1280 bytes. These mechanisms are supported in the latest 9.0 version of the widely-used BIND software [Con]. We checked the sizes of DNS response packets for the 164K unique domain names collected from the three logs in Table 5.1. The CDF for the response message size for the unique names as

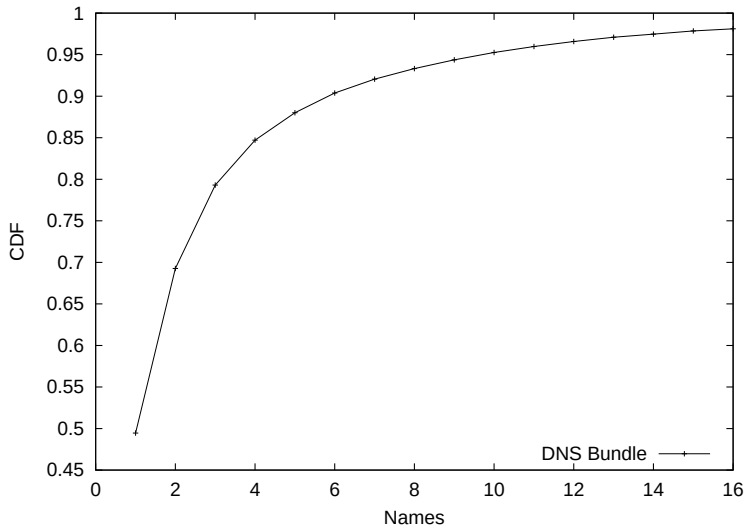


Figure 5.5: CDF of Size of DNS Bundles

well as for message sizes based on access patterns are shown in Figure 5.6. With respect to the trace-based statistic, most responses are 100-300 bytes, which affords 200-400 remaining bytes if we use the traditional 512B limit and many more bytes if we use the limit for extended DNS.

Given the available room, we examined the number of additional type "A" records that can be piggybacked on a response. If we consider type "A" RRs in IPv4, the size of all its fields are fixed except the name. While a domain name can be long, it is not necessary to put the full name in that field. DNS provides a mechanism that enables domain names to share their common suffix. Using the same trace-based statistics, we observe that over 90% of names have a first distinguishing label (excluding "www") less than 10 characters while the median and average are between 4 and 5. Putting those statistics together, the length for a piggybacked record is

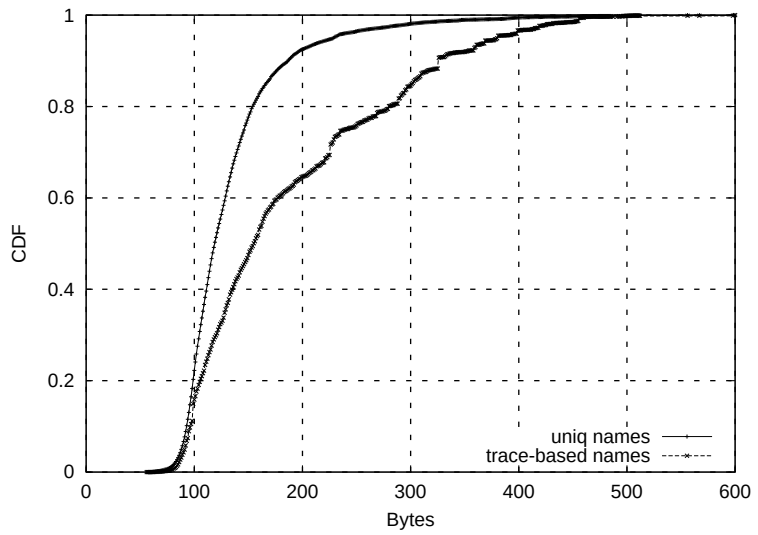


Figure 5.6: CDF of Sizes for DNS Response Packets on a Unique Name and a Trace-Based Set

likely between 18 and 27 bytes (14 bytes for all fields with fixed length, 2 bytes for the pointer to the common suffix, 1 byte for the length count for the first label, and 1-10 bytes for the first label itself). With available space of 200-400 bytes, the total RRs that can be piggybacked are between 7 (200/27) and 22 (400/18). For extended DNS, the range is between 36 and 65.

We considered the situation when one domain name maps to multiple IP addresses, which requires multiple RRs for one name. We find that over 90% of domain names have less than five associated IP addresses while 72% have only one or two. Taking this factor into account, the total names that can be piggybacked are 1-22 for the traditional DNS length and 7-65 for the extended DNS length.

5.6.3 Piggyback Policies

The previous two sets of results indicate that a sizeable percentage of the records that could be piggybacked will fit in the additional space of a DNS response. In cases where there are more potential names than can be piggybacked, an ADNS needs to have a policy to decide which names to include. In addition to the *ideal* policy, which we described in Section 5.5, we define two practical policies: *Most Frequently Queried (MFQ) First* and *Most Related Query (MRQ) First*. The former policy gives preference to piggybacking names that are popular in the zone independent of the current request, while the latter policy gives preference to piggybacking names that are most related to the current query. These policies are described in more detail as follows.

MFQ(n): The ADNS selects up to n names in the order of their requested frequencies. For this policy, the ADNS needs to track query frequencies for each name in its zone and maintain them in a Frequency Ordered List (FOL).

MRQ(n, r): The ADNS selects names in the order of their relevancy to the current query. A Relevancy Ordered List (ROL) is maintained for each name. ROL(a) denotes the relevancy list for domain name "a". The MRQ policy chooses names from the ROL list with a relevancy greater than r for the current query up to the bound of n. If there is still remaining space then names from the FOL are added.

```
1: twdns-01.ns.aol.com. : #zone cnn.com, identified by its first ADNS
2:  i.cnn.net(4379) www.cnn.com(1494) sportsillustrated.cnn.com(588) money.cnn.com(263) ... fyi.cnn.com(1) # FOL
3:  www.cnn.com(723) i.cnn.net(0.78) money.cnn.com(0.07) ... # ROL(www.cnn.com)
4:  sportsillustrated.cnn.com(271) i.cnn.net(0.42) www.cnn.com(0.09)  ... #ROL(sportsillustrated.cnn.com )
… …
51: www.cnnfn.com(7) i.cnn.net(1.00) www.cnn.com(0.29) money.cnn.com(0.14)  …  #ROL(www.cnnfn.com)
… …
```

Figure 5.7: FOL and ROLs for zone "cnn.com."

Figure 5.7 shows an example of a FOL and ROLs for the zone "cnn.com" based on queries from the WPI DNS log. The first line contains the name of first ADNS (in sorted order) for the zone "cnn.com." The second line is the FOL for the zone and has all names in the order of their query frequency. All subsequent lines are the ROLs for each name. The first element on each line is the name and the remaining elements are its related names in the order of their relevancy values.

5.6.4 Maintenance of Information

Each ADNS must maintain data structures as shown in Figure 5.7 to support piggybacking of related names. In the combination of the logs in Table 5.1, we observed the maximal number of names in a zone is 1650 and the maximal number of ROLs is 627.

The FOL and ROLs can either be set up manually by administrators who know the internal connections among names, or by tracking query patterns. The example shown in Figure 5.7 is generated by analyzing the query patterns for the zone "cnn.com." The FOL is created by counting queries to each name. For generating ROLs, we group queries from the

same client to the same ADNS that occur within a short period of time (5 minutes in our experiment). Whenever a name happens to be the first query in a group, the counter for its corresponding ROL is increased by 1. For all other names (after removing duplicates) in the group, each is counted once in the ROL for the first query. The relevancy value from name "a" to name "b" is calculated by dividing the counter of "b" in ROL("a") by the counter of ROL("a"). For instance, in Figure 5.7, line 3 is the ROL for query "www.cnn.com". The following number "723" is the count for the ROL and indicates there are 723 times "www.cnn.com" is the first query in a group. The number "0.78" following "i.cnn.net" is the relevancy value from "www.cnn.com" to "i.cnn.net", which indicates out of 723, 78% of times "i.cnn.net" follows "www.cnn.com". The relationship table is created based only on the WPI DNS trace. In general, it is expected that a server could create more accurate lists based on a larger number of client users. The internal relationships and access patterns between these domain names are not expected to change in the time scale of hours so these relevancy tables can be computed offline or when the ADNS server is not busy.

5.7 Evaluation

5.7.1 Methodology

We evaluate the PRN approach by trace-drive simulations of the ideal policy as described in Section 5.5 as well as the MRQ and MFQ policies described in Section 5.6.3. We use the relative decrease in the cache miss percentage as the metric to evaluate each policy. The regular policy is used as

the baseline to compare effects of other policies with all results shown as the relative decrease in misses compared to the total number of first-seen (FS) and previously-seen (PS) misses for the regular DNS policy. Results for the ideal policy from Section 5.5 are shown for reference.

We studied the MFQ and MRQ policies with fixed upper bounds. We choose 5 and 15 as two upper bounds based on results from Section 5.6. In addition to these bounds, the MRQ policy is tested with relevancy values of 0.5 and 0 for a total of four MRQ policy combinations. Note that relevancy bound equal to 0 means the relevancy value should be bigger than 0, hence a qualified name must have some relevancy, even if weak.

5.7.2 Results

We used the first half of a log to generate relevancy tables and conducted the simulation on the second half of the log beginning with an empty DNS cache. The results are shown in Figure 5.8 and Figure 5.9 for the WPI and RTP logs respectively. Simulation on the SJ log produced similar results as the RTP log and they are not shown here. In the figures, each set of bars corresponds to a policy. The first bar in a set shows the relative (to the total misses for the regular DNS approach) decrease of first-seen misses, the second bar indicates the relative decrease of previously-seen misses, and the third bar is the relative decrease of the total misses.

The results show that both the FS misses and PS misses are significantly reduced when any piggyback policy is in use. The MRQ and MFQ policies reduce the total misses in the range from 25% to close to 40%. The results are significant because they are obtained by also reducing the number of

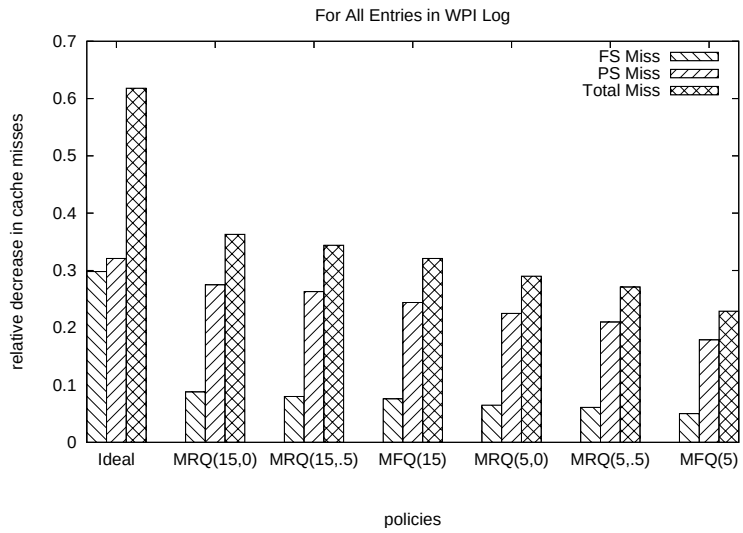


Figure 5.8: Relative Decrease in Cache Misses Over Different Policies on WPI Log

queries by the same amount.

In terms of the policies, MRQ policies consistently outperform MFQ policies when they have same bound constraints. Among MRQ policies, those having a smaller relevancy bound perform better. As MFQ(n) is similar to MRQ($n, 1$), we can summarize the performance relationship among the policies as $MFQ(n) < MRQ(n, .5) < MRQ(n, 0)$. These results indicate names with relevancy, even weak, should be given higher preference than simply piggybacking popular names. Increasing the bound helps reduce cache misses, but even the smaller bound results in a 25% reduction in cache misses.

The same methodology is used for the RTP log with the results shown in Figure 5.9. The relative decrease in cache misses for this log varies between

25% and 35% with similar variation between the policies as we found with the WPI log.

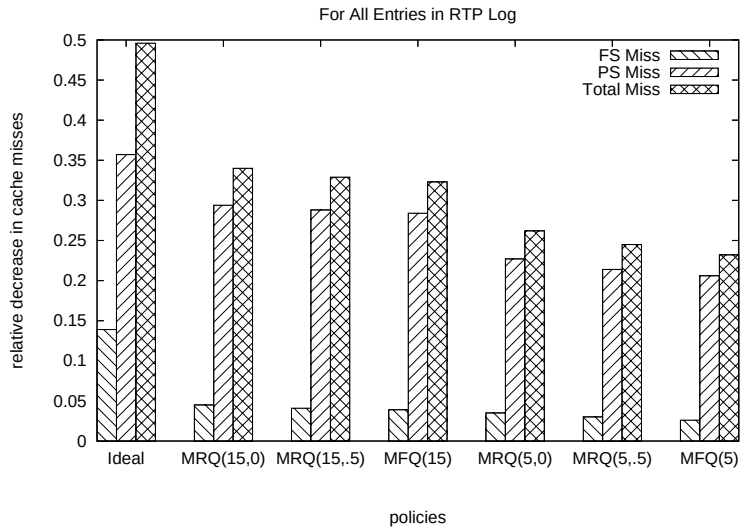


Figure 5.9: Relative Decrease in Cache Misses over Different Policies on RTP Log

5.7.3 Results for Short ATTLs

As a means to test the PRN approach for resource records with relatively short ATTLs, we filtered the log for queries to servers whose resolution have ATTLs of 30 minutes or less. This filter removed roughly half of the original DNS requests. These records must be requested more frequently by a LDNS and we hypothesize that the PRN approach would be relatively more effective at reducing the number of cache misses. Results for this analysis for the WPI log are shown in Figure 5.10 where the total miss rate for regular DNS is 32% as compared to 26% in Figure 5.3. The results in

Figure 5.10 show relative decreases in cache misses from nearly 30% to over 40%.

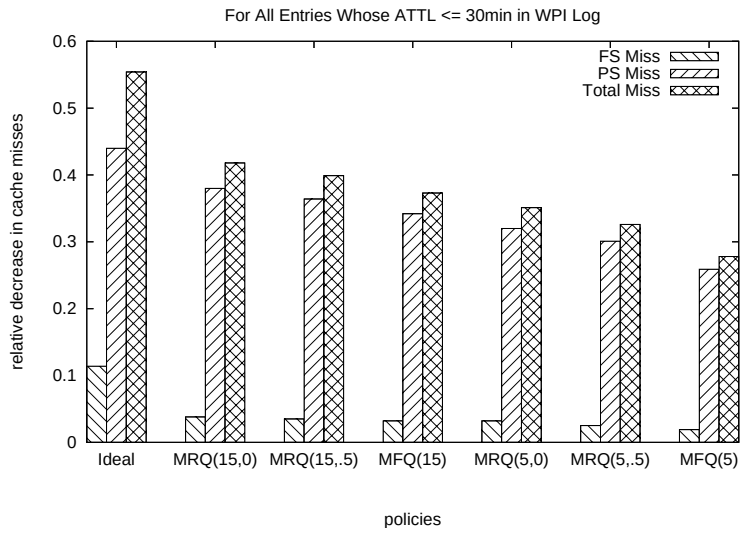


Figure 5.10: Relative Decrease in Cache Misses over Different Policies on WPI Log Entries with ATTL $\leq$ 30min.

We pushed this analysis further and filtered the log to include only entries with an ATTL of 5 minutes or less. This filter removed roughly 80% of the log entries with 46% of requests for these entries resulting in a cache miss. As shown in Figure 5.11, the PRN policies reduce the cache miss rate by over 40%. We found a similar tone of results when we did the same analysis for the RTP log. The results indicate this approach is more useful as the ATTLs grow shorter in duration.

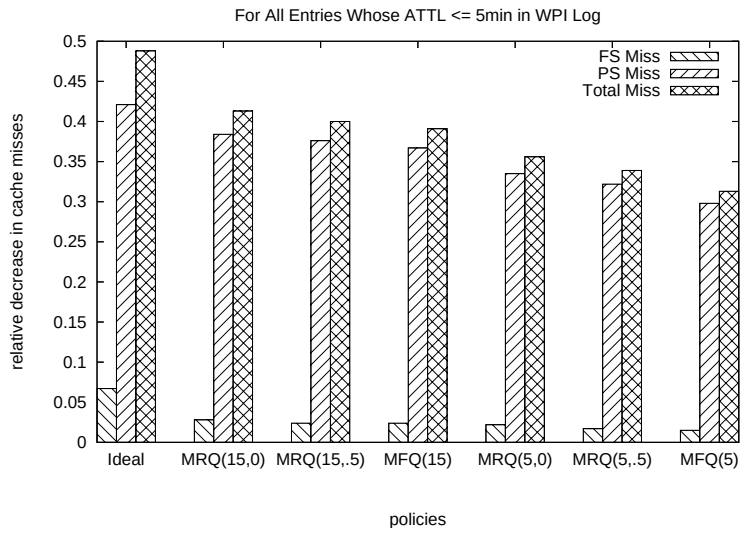


Figure 5.11: Relative Decrease in Cache Misses over Different Policies on WPI Log Entries with ATTL $\leq$ 5min.

5.7.4 Results for Total DNS Queries

Another direction we explored was the total number of DNS queries reduced by our approach. This avenue of exploration is relevant because while the PRN approach reduces DNS query traffic to ADNSs, it has little effect on traffic to root and gTLD servers. The previous results treat all cache misses as incurring the same cost when in fact the first time a LDNS encounters a domain name such as *x.foo.com* it must first find the ADNS for *foo.com*, which may involve contacting a root server as well as a *.com* gTLD server before the query is sent to the ADNS for *foo.com*. A subsequent access to *y.foo.com* would only require a query be sent to the ADNS for *foo.com* assuming the information about the ADNS is still fresh.

To model the situation where multiple DNS queries may be needed to

resolve a domain name we obtained the ATTL for all ADNSs in the WPI log. We then reran our simulation on the WPI log to determine the relative decrease of not only requests to the ADNS, but the decrease for all DNS requests, which include those to obtain the authority for a zone. We did ignore queries to root name servers, which are relatively small in number. The results are shown in Figure 5.12.

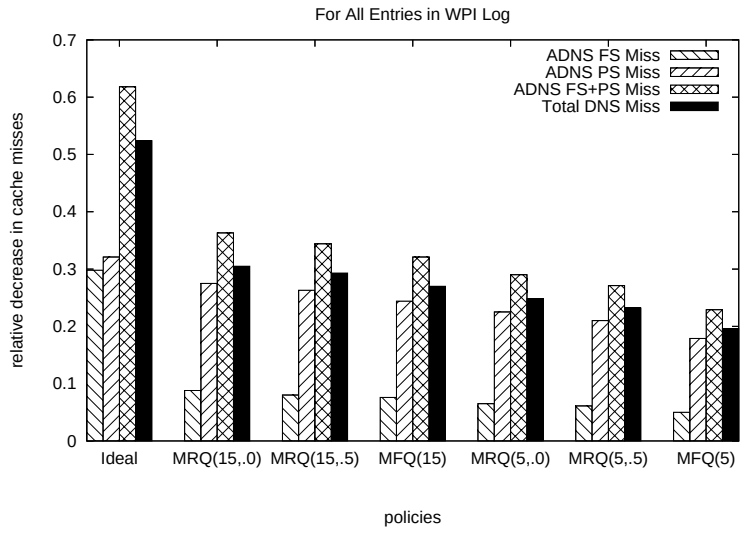


Figure 5.12: Relative Decrease in All DNS Cache Misses Over Different Policies on WPI Log

The results show that the additional DNS queries generated to obtain the authority for a zone lower the relative decrease in cache misses by less than 5%. This small reduction indicates that the number of queries generated to gTLD servers is much less than the number of queries sent directly to the ADNSs (about 20%) because the ATTLs for "A" records are generally smaller than those for records of ADNSs. Figure 5.12 shows that the PRN

approach reduces the total number of DNS requests by 20-35%.

To better understand the costs of query to a gTLD server versus an ADNS we used the `dig` DNS client from WPI to measure the respective times. Using the names from the WPI log we found a mean response time of 47ms for queries to gTLD servers and a mean of 145ms for queries to the ADNSs. For queries from a home DSL client we found queries to gTLD servers take 63ms on average versus an average of 142ms for queries to the ADNSs. These results, along with the simulation results, indicate that the requests to ADNSs are the dominant DNS costs so that an approach such as PRN does yield significant cost savings.

5.8 Comparison and Combination with Other Approaches

Our final analysis was to compare the performance between our approach and others proposed to reduce cache misses. Because our approach is compatible with the others, a combination with these approaches is possible. We evaluate these hybrid approaches on all the three logs with results for the WPI log shown.

5.8.1 Performance Comparison Among Approaches

The proactive caching approach proposed in [CK01] has several policies. Among them, R-LFU is one of the better and more straightforward policies. We implemented R-LFU(r) for comparison purposes. The renewal using piggyback (RUP) approach proposed in [JC03] also has several policies. We implemented RUP-MFU, which performs best among all practical

approaches. Among our PRN policies, MRQ performs better than MFQ. We choose MRQ(15,0) for the comparisons between approaches. We refer it as PRN-MRQ(15,0).

Figure 5.13 shows the relative decrease in cache miss percentages for the three approaches relative to normal DNS. The reduction in total cache misses is close for all the three approaches. When considering FS misses and PS misses separately, RUP-MFU and R-LFU policies behave almost the same, where FS misses are untouched and the reduction rates for PS misses are close. While PRN-MRQ does not reduce PS misses as much as the other two, its reduction on FS misses compensates for the difference. Despite the fact that the performance gains among the three approaches are similar, their costs are different. For the variation of the R-LFU policy we studied, it introduces 56% more queries and responses than normal DNS. The PRN-MRQ and RUP-MFU policies do not produce additional queries. Instead, by reducing the total misses, the total queries and responses are reduced as well.

5.8.2 Combination of PRN and RUP

In Figure 5.13 we observe that PRN-MRQ reduces more FS misses while RUP-MFU reduces more PS misses. This result encourages us to consider the possibility of combining the two approaches. Both approaches use piggybacking, but one makes the decision on the server side while the other does on the LDNS client side.

To combine these policies we define a new policy called "piggyback related names with client hint first" (PRN-CHF), where the client DNS server

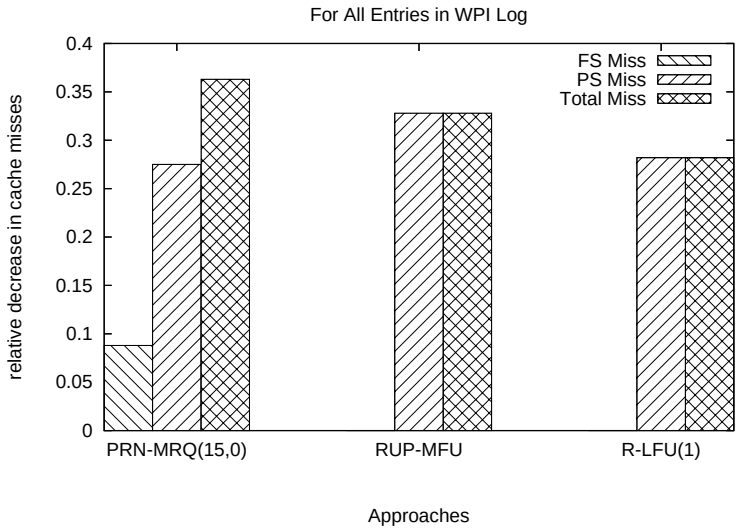


Figure 5.13: Performance Comparison among Approaches on WPI Log

piggybacks its stale names in the query message and the ADNS uses these hints as well as its own relevancy table. The policy is described as:

> PRN-CHF(n, r): The total number of names that can be piggybacked is bounded by n, but instead of first looking at the corresponding ROL, the ADNS gives priority to the names piggybacked in the query message. If there is still extra space left, the ROL and FOL are checked in turn.

We show the performance of PRN-CHF and its two component approaches in Figure 5.14. As we expected, PRN-CHF has the same FS miss rate as PRN-MRQ and the same PS miss rate as RUP-MFU, so it performs the best among the three.

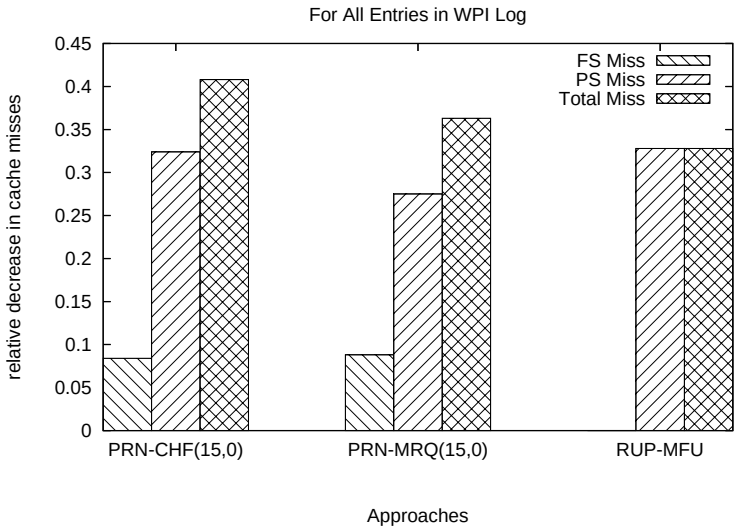


Figure 5.14: Performance Comparison among PRN-CHF and its Two Component Approaches on WPI log

5.8.3 Combination of PRN and R-LFU

We also studied the combination of these policies with active renewal. As the R-LFU approach is initiated by the LDNS cache and the PRN approach is initiated by an ADNS, the two approaches can complement each other.

We show performance of the various hybrid approaches along with R-LFU in Figure 5.15. In order to distinguish our original PRN approaches from their hybrid versions with R-LFU, we refer to those three hybrid approaches with a prefix "R-" to their original names. As with the R-LFU approach, each approach is tested with different aggressiveness in prefetching. As aggressiveness increases, the cache misses are further reduced, but more queries are generated.

The hybrid approaches show significant performance gains in Figure 5.15

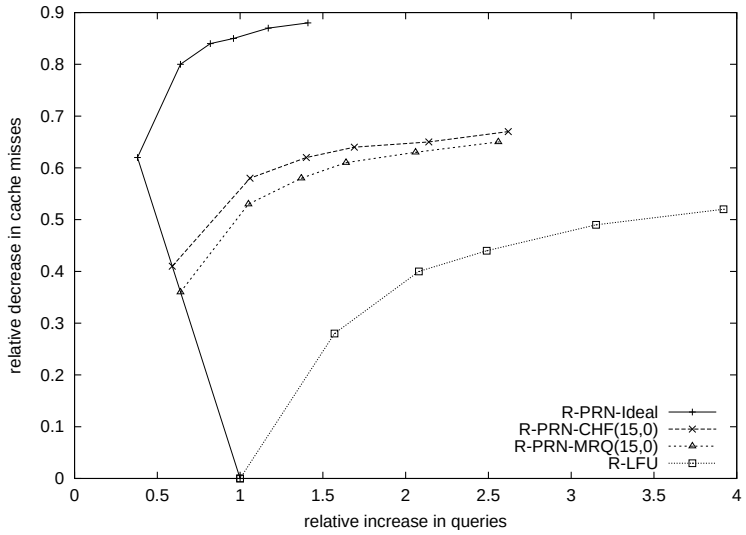


Figure 5.15: Performance for Hybrid Approaches on WPI Log

compared with either R-LFU or their original PRN approaches. With about the same number of queries, R-PRN-MRQ performs much better than R-LFU. For instance, having 1.5 times queries as the regular approach, R-PRN-MRQ reduces 54% of total misses while R-LFU reduces 28%. R-PRN-CHF performs slightly better than R-PRN-MRQ as it is the combination of the PRN, RUP and R-LFU approaches. Renewal also benefits the original PRN approach at the expense of more queries.

5.9 Summary

This work is motivated by research on studying the relationships among network data flows. We found many cases where a local DNS server sends multiple DNS queries to the same authoritative DNS server within a short

period of time. If the ADNS can predict these near-future queries once it receives the first one then it can send answers for all of them with the first response. We call this the piggybacking related names (PRN) approach. It helps reduce local cache misses and therefore reduces user-perceived DNS lookup latency. By piggybacking multiple answers in one response packet, the total queries and responses are also reduced, which alleviates the workload on both LDNSs and ADNSs.

Compared with other approaches that also address improving local cache hit rate, our approach is novel. We explictly use the relationships among queries and allow an ADNS to push resolutions for predicted names to the LDNS. The PRN approach reduces both first-seen misses as well as previously-seen misses while other approaches reduce just the latter. The cost of PRN is also low as it reduces the number of query and response packets while requiring no changes to the DNS protocol.

Trace-base simulations show more than 50% of cache misses can be reduced if prediction is perfect and response packet space is plentiful. Realistic policies, using frequency and relevancy data for an ADNS, reduce cache misses by 25-40% and all DNS traffic by 20-35%. These percentages improve if we focus the policies on resource records with smaller ATTLs. We also show improved performance by combining the PRN approach with renewal-based approaches to create hybrid approaches that perform significantly better than their component approaches.

In conjunction with this work we also did a study on current DNS performance for 20 locations in the United States. The outcome of this study is that the current average DNS latency is generally in the range of 200-300ms,

but range from 500ms to multiple seconds if we look at the 95% response time. The reduced cache misses for the PRN approach will be reflected in improved response latency and timeout performance.

Chapter 6

Data Piggybacking

In the previous chapter, we show an example of how a particular relationship between domain names is used to improve DNS performance. It takes advantage of available packet space provided in the first DNS response message. In this chapter, we look at exploiting packet space in a more general way by using acknowledge (ACK) packets on the reverse direction of a TCP data flow. We use this example to show how available packet space within flows is used to improve transmission efficiency.

TCP is a widely used protocol on the Internet designed for reliable bidirectional data transfer. However, large numbers of TCP connections show an asymmetric traffic pattern where significantly more data is sent in one direction than the other. It is a common scenario with applications such as HTTP or bulk data transfer that a client initiates a request to a server and then the data flow is entirely from the server to the client. Traffic in the reverse direction is only TCP ACK packets to acknowledge receipt of the downloaded data. In addition, packets used for TCP connection estab-

lishment and teardown also have minimal TCP packet size and carry no user data. For convenience, we call all TCP packets that carry only TCP/IP headers (and possibly some TCP options) *ACK-only* packets.

The large number of ACK-only TCP packets on the Internet is well documented. Results in [TMW97] show a trimodal distribution of TCP packet sizes where the size of ACK-only packets (40 bytes with no TCP options) is one of the modes. Statistics in [MC00] show just under 50% of TCP packets are of TCP header size. More recently, various traces from the IP Monitoring Project taken at monitoring points in the SprintLink IP backbone in February 2004 show 40-70% of packets in individual traces are of TCP header size [Spr04]. An ongoing study [SPH05] also shows that over 40% of packets are 40 bytes for traffic collected at five different network points, including Los Nettos, a USC Internet2 connection, and three connections monitored by NLANR during December 2004 to October 2005 period. In addition, our packet size study conducted in Chapter 3 shows that over 60% of packets are of TCP header size in the upstream direction (from client to server).

Our approach is to piggyback application-level data onto ACK packets sent in the reverse channel. Normally TCP piggybacks acknowledgment information onto the data packets it sends. In this work we explore the potential for a mechanism to piggyback data only when an ACK packet would normally be generated. If a mechanism was available for applications to send reverse-channel data without generating additional network packets then client receivers of data could upload feedback to the server providers of the data without incurring new connections or generating ad-

ditional traffic. Such a mechanism could also be used by peers in a peer-to-peer environment where the transfer of desired content from peer A to peer B could simultaneously support the exchange of useful content via piggybacked transfer from B to A. We discuss the piggybacking data mechanism, experiment setup, and evaluation results in the following sections.

6.1 Mechanism

The TCP protocol allows data transfer in both directions of a connection with ACKs for data packets received in the forward direction piggybacked onto data transferred in the reverse direction. However because many connections primarily transfer data in only one direction many ACK-only packets are generated by the receiver. The key idea of our work is to invert the traditional TCP mechanism and piggyback data onto packets carrying needed ACK information. This approach creates a clear primary and secondary direction of data flow within a TCP connection.

This approach is interesting to explore because it allows data transfer to occur in both directions while being potentially more efficient in the number of packets generated for data transfer in the reverse direction. Potential applications of this approach include asymmetric connections where previous work shows that less bandwidth in the reverse direction impacts performance for forward data transfer [BPFS02]. Other work [KVR98] has shown that bidirectional traffic can cause reduced throughput due to undesired interaction effects such as ACK compression [ZSC91].

Peer-to-peer (p2p) applications can also be written to take advantage

of reverse ACK traffic where the transfer of content from peer A to peer B could simultaneously support the transfer of useful content from peer B to peer A. Incentives in a p2p application such as BitTorrent encourage clients to exchange data tit-for-tat in both directions [Coh03], which could be done more efficiently with our approach. Others have proposed the idea of generalizing BitTorrent with a Data Exchange Market [Tur05]. Our approach could be used with this idea as well.

Our approach does require a new mechanism to be supported by TCP with modifications to client and servers. To illustrate, Figure 6.1 shows the core code for a standard data transfer from a server to a client. The server continually sends data in a buffer (buf) to a socket (s) while the client reads data from its socket into a buffer and processes it.

```
// Client                          // Server
while (not done) {                 while (not done) {
  recv from s into buf;              put data into buf;
  process buf data;                  send buf to s;
}                                  }
```

Figure 6.1: Standard Core Client and Server Code

Figure 6.2 shows the modified code using a new data piggybacking TCP option, which causes the TCP implementation to only send buffered data if a packet is generated to ACK data received. In Figure 6.2 the client checks if reverse direction send buffer space is available and if so then sends data to the socket (s). Otherwise the client works just as the standard case. The server requires fewer modifications as it simply checks the availability of

input data on the socket, using a call such as `select()`, and if available it receives and processes that data. Because each loop is driven by the data being sent in the forward direction a mechanism is needed for the client to query how much of the buffered data has actually been sent. Depending on the application, the client may need to finish sending any unsent data via the traditional mechanism or simply terminate the connection if the data do not need to be sent.

```
// Client                          // Server
turn on data piggyback;            while (not done) {
while (not done) {                   if (revdata avail) {
  recv from s into buf;                recv from s into rbuf;
  if (send buf space)                  process rbuf data;
    send revdata to s;               }
  process buf data;                  put data into buf;
}                                    send buf to s;
                                    }
```

Figure 6.2: Modified Core Client and Server Code

A primary issue with this approach is how much data can be piggybacked onto an ACK packet. In a bandwidth constrained environment trying to piggyback too much data could have a negative effect of forward traffic. This issue is examined in our testing. In our testing we did not modify the TCP implementation for this initial work, but used the existing TCP implementation with the code shown in Figure 6.2. The result is that each time data is received from the socket in the loop, we send reverse data to the socket if buffer space is available. The amount of reverse data sent is a parameter of each experiment. This approximated approach does not

guarantee that reverse data and ACKs are sent together, but with a client buffer big enough to receive a full packet we observe that generally each application-level receive corresponds to a packet reception thus the client generally sends one packet for each received. Ideally, we could have a kernel-level support to ensure that each data transmission matches an ACK packet. However, we use the user-level approach to quickly test the concept of data piggybacking. The experiments over different paths do show that the results using the approximated approach are close to those we calculate under the ideal situation where the kernel-level support would be available. With only a user-level modification, we are able to test the data piggybacking method widely as no privilege to change TCP/IP kernel is required.

6.2 Testing Environment and Methodology

We tested the piggybacking data approach over network connections between our home institution of WPI, on the east coast of the U.S., and seven endpoints with various RTT and throughput connectivity values as summarized in Table 6.1. The RTTs were measured by calculating the time interval between the SYN and SYN ACK packets when TCP connections were established. The throughput was measured at the application level, which is the ratio of the data size over the time used to transmit the data (more strictly speaking, we measured goodput). The RTT and throughput values are representative of those obtained during testing, although some variation in packet loss occurred, which is noted as appropriate. Four of

the links to institutions in California and Georgia in the U.S. as well as to Italy and the Netherlands (NL) show relatively good bandwidth in both directions, although as shown in Table 6.1 demonstrate asymmetric throughput. Further investigation of this throughput asymmetry found that for the California to WPI path the advertised receiver window was 64K bytes by Linux on the WPI side while it was only 32K bytes by the Linux version running in California. For the other three links, the asymmetric throughput is primarily due to higher packet loss in one direction than the other.

The three other links in Table 6.1 do exhibit asymmetric bandwidth. These links include local machines accessing WPI via DSL and cable modem as well as one in the Netherlands connected via DSL. All seven of these links were used for data piggyback tests described in the following section while just the Calif/WPI, Italy/WPI and WPI/Local DSL links were used for the enhanced ACK tests described in Section 7.2 because tcpdump was needed on both sides of the link to capture packets. Unless noted, all tests were run using Linux TCP implementations (we used Linux kernel version 2.4 and 2.6 in the experiments).

The question for the data piggybacking approach is to understand how much data can be piggybacked with ACK packets without introducing more packets and not influencing the performance of forward-channel traffic. We answer this question with a series of experiments conducted under the variety of real network conditions described in Table 6.1. For each experiment we evaluate performance and efficiency of different transmission methods, where throughput (transmitted bytes divided by transmission time) is used as the metric of performance and packet counts is used as

Table 6.1: Summary of Network Connections Used in Experiments

End Points		Thruput(KBps)		RTT
A	B	A to B	B to A	(ms)
WPI	Calif	350	600	80
WPI	Italy	400	200	120
WPI	Georgia	1100	600	30
WPI	NL	520	150	90
WPI	NL DSL	280	90	80
WPI	Local DSL	190	27	40
WPI	Local Cable	350	10	40

the metric of efficiency.

As a standard, we used the transfer of a 1MB file for all testing. We deliberately chose this size as its transfer normally takes around 700 packets, which is large enough to minimize throughput effects of slow start, but small enough for a reasonable experimental time.

We evaluated the approach described in Section 6.1 where a server sends a 1M byte file in the forward direction to a client. As shown in Figure 6.2, each time the client reads some data it sends data of a particular size to the server if buffer space is available. Ideally the TCP implementation on the client-side only sends the data when it would normally generate an ACK, but given that we have not modified the TCP implementation the reverse data sent is not exactly matched with the sending of ACKs. However, because the client application code sends data immediately after it receives data from the server, the sending of the data is roughly matched with the sending of ACKs (that are generated for the received data). As part

of the experiment we control how much data is sent in the reverse direction for each packet received. In an ideal implementation, this amount of data would be piggybacked with each ACK. Idealized values for throughput and the number of packets are shown in the results for each size.

In addition, we evaluated two control approaches: In the first, labeled "2Con" in the results, the algorithm in Figure 6.1 is used by two identical and separate processes on each side to transfer 1MB in each direction. In the second control approach, labeled "1Con", 1MB is again transfered in each direction, but only a single connection is used for the transfer so that ACKs may be piggybacked on reverse data traffic. The identical endpoints are written so that they do not needlessly block waiting to send or receive data.

For each network condition, experiments are conducted with each of the three approaches, where the size of piggyback data sent in the reverse direction varies from zero (i.e. no piggyback) to 1500 bytes. Note that the size of 1500 bytes is over the size of the MSS (1460 in this experiment). It is intended to check the effects when the piggybacked data is more than the MSS. The maximum piggyback size tested within the MSS is 1400 bytes in these experiments. Results are reported based on five runs for each case, although in cases where the loss rate is generally over 1%, 10 runs are conducted to mitigate fluctuations in throughput.

6.3 Results

Tests for all links in Table 6.1 were made as described in the previous section. Rather than show all results, this section shows results across a representative set of network conditions. The first set of results are shown in Figure 6.3 for the connection from the California site to WPI. The first graph shows throughput results and the second graph shows packet count results. In each figure, the curves on the left side are for the piggyback approach, where one curve represents the download direction and the other curve represents the upload direction. There are two sets of bars on the right side of each figure. The rightmost set of bars are for the 1Con approach where one network connection is used to download and upload a 1MB file. The set of bars to its left are for the approach where two individual connections are used. In each set of bars, the left one represents the download direction and the right one represents upload direction.

For the Down and Up piggyback curves, the measured throughput and packet count results are shown for piggyback sizes of 0 (no piggybacking), 50, 100, 200, 300, 400, 700, 1000, 1400 and 1500 bytes. The Ideal throughput results are obtained by multiplying the number of ACKs observed in the no piggybacking case by the piggyback size, constrained by the maximum observed reverse throughput.

The results show that the downlink throughput is unaffected by the piggyback size while the uplink throughput rises to a level comparable to the 2Con and 1Con approaches. The packet count graph shows the number of uplink packets to be relatively constant and fewer than the control cases.

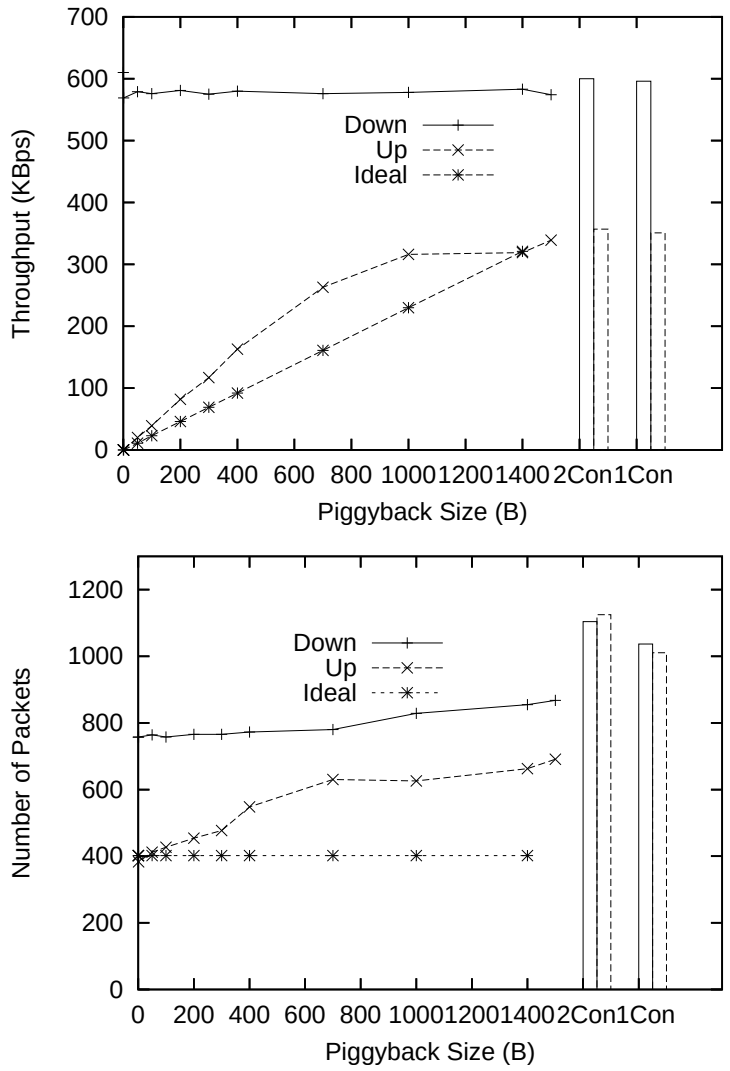


Figure 6.3: Calif to WPI (less than 1% packet loss)

More important, the uplink packet count is significantly less that the two control cases with comparable throughput for the largest piggyback sizes. The results show that for piggyback sizes of a few hundred bytes, the existing TCP implementation does not generate many more packets than the ideal implementation.

Figure 6.4 shows another network connection within the U.S. with minimal packet loss, but a shorter RTT. The results are comparable to Figure 6.3 with similar throughput as the control cases while generating significantly fewer packets than the control cases. The throughput of the reverse direction flattens after the piggyback size goes over 1000 bytes because of some packet loss. When packet loss occurs no new data is read by the client and thus no new data is sent as shown in the code of Figure 6.2. With an ideal TCP implementation, an additional check to ensure the send buffer always has data could be added to the beginning of the loop to make sure data is still available for piggybacking onto duplicate ACKs and hence provide better reverse throughput.

Figure 6.5 shows a longer network connection with a packet loss rate of 1-5% observed during testing. The effect due to lost packets is even more pronounced in this test as the Up throughput flattens out beginning with piggyback sizes of 400 bytes while an Ideal implementation would allow data to be piggybacked on duplicate ACKs generated because of errors.

Finally Figure 6.6 shows results for a forward network connection from WPI to a host connected locally via DSL. As the throughput results show, the downlink throughput is significantly affected once the uplink capacity of approximately 40KBps is reached. The control approaches show how

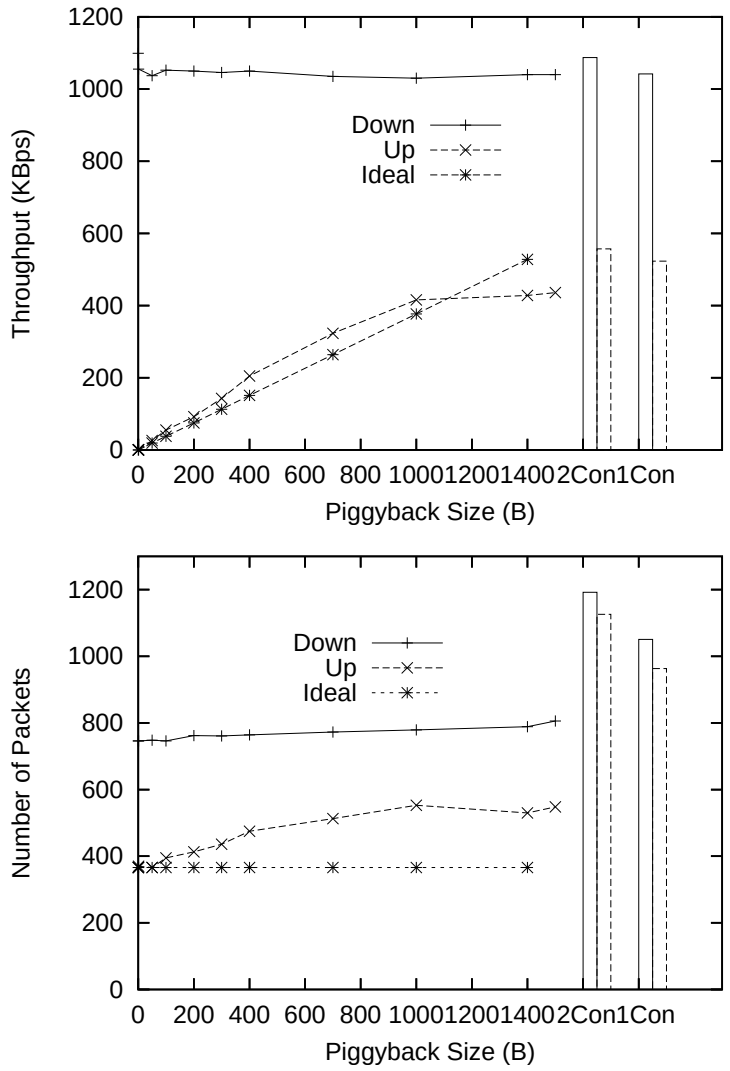


Figure 6.4: WPI to Georgia (around 1% packet loss from GA to WPI)

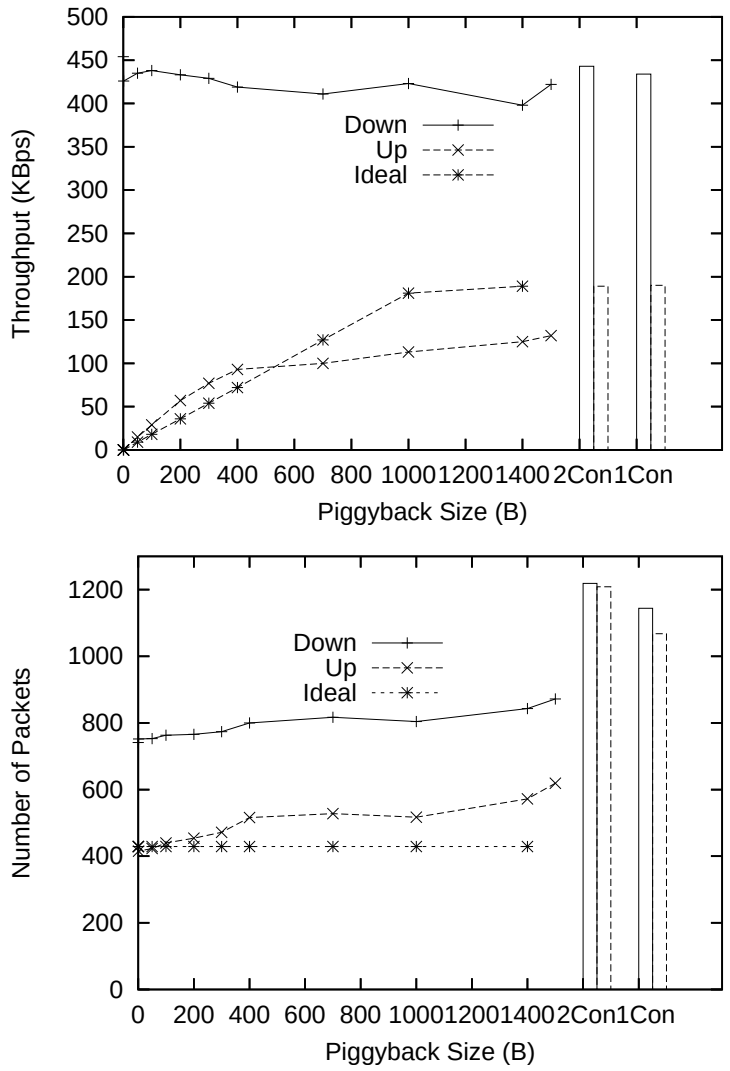


Figure 6.5: WPI to Italy (1% to 5% packet loss from IT to WPI)

saturation of the uplink negatively affects both directions. For this asymmetric network, these results do show that our approach can be effective for uploading more modest amounts of data without increasing the number of uplink packets and without impacting the downlink throughput.

6.4 Observations

Results from the previous experiments lead to a number of observations about the potential benefits of the data piggybacking. We found that the reverse channel throughput can match the effective reverse bandwidth limit without negative effects on forward channel throughput. Second, the number of reverse channel packets generated to achieve this throughput is significantly less than a simple bidirectional transfer over one connection or two independent connections. Even in the case of an application-only approach with no TCP implementation support there is a reduction in the number of reverse-channel packets. Third, even in the case of asymmetric links such as a home DSL connection, data piggyback sizes up to a few hundred bytes per packet can be supported in either an application-only or TCP-level implementation without reducing forward-channel throughput nor having an appreciable effect on the number of reverse-channel packets.

Having data piggybacked by ACK packets not only saves the total number of packets, but also the total number of bytes as less packet headers are generated. A drawback of this scheme is that it requires availability of ACK packets in order to send data on the reverse direction. For the situations when data needs to be sent immediately but no available ACK packets,

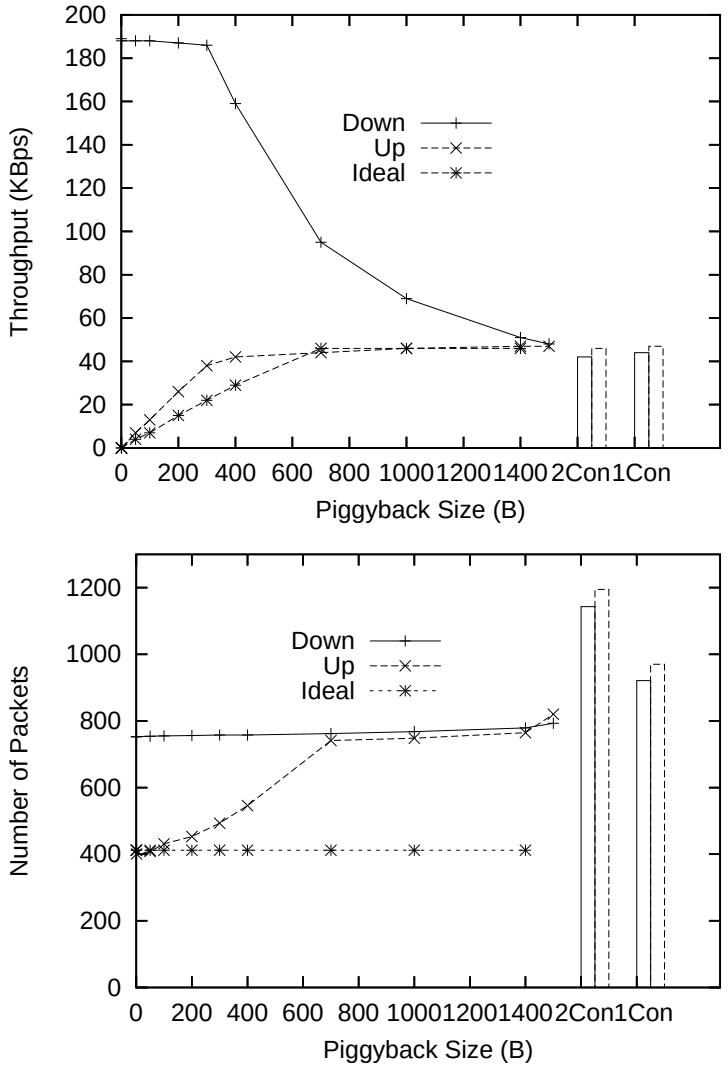


Figure 6.6: WPI to Local DSL Home (less than 1% packet loss)

separate packets can be generated as the current TCP does. The current socket APIs should be extended to allow applications to indicate whether they want this feature enabled or not.

An interesting question about the approach is whether it improves or exacerbates problems that occur due to dropped or out-of-order packets. In the case of data piggybacking and forward-direction congestion, duplicate ACKs in the reverse direction provide more opportunities for piggybacking. However, too much data piggybacked in the reverse direction could cause congestion and negatively affect forward throughput.

6.5 Summary

In this work we have discussed and evaluated the data piggybacking approach, which provides packet-efficient throughput in the reverse direction of a connection without sacrificing forward throughput. This work is motivated by the observation of lots of ACK-only packets on the current Internet. These packets can provide a certain transmission capacity on the reverse channel of a TCP connection without introducing extra packets. This approach creates a clear primary and secondary direction of data flow within a TCP connection. Data are piggybacked on the reverse channel only when an ACK packet would normally be generated.

By using a user-level modification to a regular client and server code, we are able to approximate the effect of letting ACKs to piggyback user data on the reverse channel. Results show that the reverse channel throughput can match the effective reverse bandwidth limit without negative ef-

fects on forward channel throughput. The number of reverse channel packets generated to achieve this throughput is significantly less than a simple bidirectional transfer over one connection or two independent connections. Even in the case of asymmetric links such as a home DSL connection, data piggyback sizes up to a few hundred bytes per packet can be supported in either an application-only or TCP-level implementation without reducing forward-channel throughput nor having an appreciable effect on the number of reverse-channel packets.

Chapter 7

TCP Enhanced ACK

We have looked at two particular approaches that exploit flow relationships and available packet space within flows in the previous two chapters. The approach of piggybacking related domain names directly improves performance of DNS application. The method of piggybacking data improves transmission efficiency without hurting application performance. In this chapter, we propose an approach to let a receiver provide additional control information to the sender via additional TCP header information. We use this approach as an example to show how available packet space is used to provide better quality of information for applications.

This work is also motivated by the observation of a large number of ACK-only packet on the Internet and the fact that limited control information is provided by current TCP acknowledgements. We examine the introduction of a new TCP option that provides more detailed and more complete information about the reception of data packets at the receiver compared with the existing TCP Timestamps Option [JBB92]. This infor-

mation allows a TCP sender to track the spacing between all data packets arriving at the receiver and to have complete timing information for the forward and reverse directions of the connection. The information can be used to better detect jitter and congestion in the forward direction than what is currently available.

By using this new option, TCP can also have a more accurate RTT estimation, which is not influenced by the factors of ACK delay or packet loss. If we can extend the current "tcp_info" structure [tcpb] to include this RTT information along with the delay jitter information, an application can have a better understanding of current network conditions, therefore adjusting its data generation decision more appropriately. For example, a streaming application needs to adjust its sending rate based on network conditions such as RTT, packet loss, and delay jitter. If TCP can already provide this information, the streaming application does not need its own mechanism in order to get the same set of information.

7.1 Mechanism

This approach uses a bit of the available bandwidth to enhance the contents of ACKs to provide more detailed and complete information about the reception of data packets at a receiver. The TCP Timestamps Option allows TCP implementations to calculate round-trip times (RTTs) on more than one packet per window of packets and allows time stamp echoing in either direction [JBB92]. The contents of the option, shown in Figure 7.1, include four fields: one-byte each for the kind (k) and length (l) with four-bytes

each for the packet timestamp (TSval) and echo reply timestamp (TSecr).

k	l	TSval	TSecr
1	1	4	4

Figure 7.1: TCP Timestamps Option Layout

The use of these timestamp fields is illustrated in Figure 7.2, which shows two common scenarios for how the Timestamps Option is used when an ACK is generated by the receiver. The example is illustrative and does not show all packets in a TCP connection nor does it show how the TCP option is used when packet loss occurs. The first ACK in the example is generated at time 30 in response to data segment 1 being received at time 28 with a TSval of 18. The delay between when a packet is received and the corresponding ACK generated is defined in [APS99] to be within 500ms of the arrival of unACKed packet and at least every second full-sized segment, although 200ms is a commonly used maximum time. Previous work [BG99] found that transmission of small data packets often incurred close to 200ms ACK generation delay in the presence of the TCP Nagle algorithm [Nag84].

The first ACK TSval indicates it was sent at time 30 and is an echo response to the packet with timestamp 18. The second ACK is generated at time 70 after two data segments have been successfully received. This behavior occurs with the TCP delayed ACK feature, which effectively ACKs every other packet. The TSecr value of 45 indicates the timestamp of the earliest previously unACKed packet as defined in [JBB92].

The Timestamps Option was proposed to enable the sender of a TCP connection to more frequently determine the RTT of sent data segments,

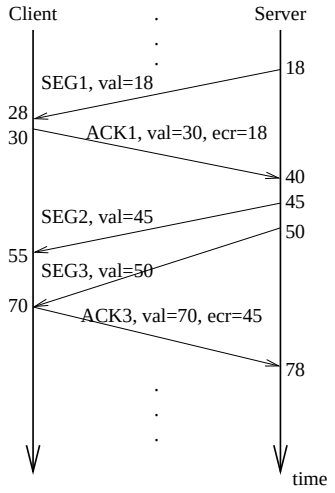


Figure 7.2: Example Usage of TCP Timestamps Option

but as the example shows the option still does not capture as much information about the nature of the connection as it could. We conjecture that more complete information about the pattern of received packets would allow better transmission of sent packets despite issues due to congestion or the delayed ACK option. It can also help to detect the effects of ACK compression when ACKs are grouped together [ZSC91].

The layout for the variable-length enhanced TCP Timestamps Option we propose is shown in Figure 7.3. It extends the existing Timestamps Option in two ways. First, the reception time for each data segment received by the receiver is returned to the sender in a TSrcv field, and second the TSecr and TSrcv values are returned for all packets received since the last ACK was sent. Given that the maximum size of a TCP option field is 40 bytes, information for up to four packets could be included, although if every other packet is ACKed then information for no more than two pack-

ets would need to be included. For the example in Figure 7.2, the second ACK would have the fields: val=70, ecr1=45, rcv1=55, ecr2=50, rcv2=70. Once sent, the receiver no longer needs to retain the information thus the amount of receiver state maintained is bounded.

k	l	TSval	TSecr1	TSrcv1	TSecr2	TSrcv2	···
1	1	4	4	4	4	4	

Figure 7.3: Enhanced TCP Timestamps Option Layout

This enhanced Timestamps Option provides three improvements relative to current TCP functionality.

1). The capability to measure one-way jitter and packet spacing in each direction. The enhanced option allows the packet spacing on the receiver side to be communicated to the sender rather than trying to use ACK spacing at the sender to approximate packet spacing.

2). The option explicitly captures the delay for generating an ACK at the receiver. While ACK generation may be immediate for full-size packets received during slow start or when the second of two full-sized packets is received, the sender does not know unless this information is recorded.

3). The option captures the delay information for all received packets. In the presence of delayed ACKs, information is lost because an ACK is only generated for every other received packet and there is no way to know when the first packet of each pair arrives at the receiver. The current Timestamps Option is intended for a round-trip measure that

includes delays, not for precise timing of each packet.

We envision the enhanced Timestamps Option to be particularly useful in asymmetric networks. Balakrishnan, et al [BPK99] describe a number of issues using TCP with asymmetric networks where low bandwidth on the reverse link causes problems for the timely arrival of ACK packets needed for the ACK-clocked nature of TCP. Variants of TCP such as TCP Vegas [BP95] and TCP Westwood [WVSG02] use rate estimation to drive when packets are sent based on the rate at which ACKs are received. However work such as [FCL01] shows that in asymmetric networks TCP Vegas does not perform well because it is using ACK rate to estimate data rate at the receiver. They describe the need to encode the arrival times at the receiver and show improved results if these data are available, but imply the TCP Timestamps Option can be used for gathering forward path flow rate without specifying details. Similarly, [AER98] uses the Timestamps Option to estimate forward trip time in a TCP connection, but this option does not account for ACK generation delay and it loses information when delayed ACKs are used. Finally, the availability of one-way jitter information allows investigation of using jitter to predict congestion loss before it occurs. As part of measurement work on audio transmission, [RASHB02] found that RTT variation can be an indicator of packet loss.

7.2 Testing Methodology

The methodology for testing the enhanced ACK Timestamp mechanism seeks to examine the measurement of connection metrics using the en-

hanced mechanism compared to using TCP with no options and the existing TCP Timestamps Option. With our enhanced Timestamps Option, the sender can learn exactly when the data packets are received on the receiver side. It allows several potential benefits:

- enables the capability to calculate packet interarrival spacing, which can be combined with packet sending spacing to determine forward direction jitter effects,
- provides the sender information about the delay for generating an ACK at the receiver, and
- allows more accurate calculation of RTTs by the sender.

We choose packet interarrival spacing, ACK generation delay and RTT as the three metrics to examine the potential benefits of using the enhanced Timestamps Option compared to currently available methods. Table 7.1 summarizes how these metrics are calculated with the enhanced Timestamps Option as well as how they are calculated with current methods.

The interarrival spacing of data at the receiver is important because this information can be combined with the spacing of sent data to monitor the queuing delays in the forward direction, which may be used to infer congestion in this direction before packet drops occur. The first metric in Table 7.1 shows how the data arrival spacing is calculated with each of the three approaches. The spacing can be explicitly calculated using our enhanced Timestamps Option. With no Timestamps Option, the sender must match an ACK with the corresponding sent data. This match is non-trivial

Table 7.1: Means to Measure Connection Metrics Using Different TCP Options

Metric	Enh. ACK Timestamp	TCP w/out Timestamp	TCP TS Option [JBB92]
Data Inter-arrival Spacing	Explicitly Calculate	Use ACK Recv Spacing	Use ACK Send Spacing
ACK Generation Delay	Explicitly Calculate	unavailable	unavailable
RTT	Recv-TSecr -ACKDelay	Recv-Send	Recv-TSecr

to do in the case of packet loss and the use of delayed ACKs. The spacing calculation also includes ACK generation delays as well as reverse direction congestion effects. The use of the current TCP Timestamps Option removes reverse direction congestion effects from the calculation, but ACK generation delays are still present. In addition, with the presence of delayed ACKs, an ACK may represent two received data packets.

As shown in Table 7.1, the delay to generate an ACK at the receiver is explicitly captured with our enhanced Timestamps Option, but unavailable using existing approaches. Variability in the ACK generation algorithm by a TCP receiver introduces variability in the ACK receiver spacing at the sender regardless of any congestion in the network between the sender and receiver.

The final row in Table 7.1 shows how the RTT is calculated using each of the three approaches. The existing TCP Timestamps Option allows more frequent and accurate RTT calculation than without use of the option. Our approach yields a yet more accurate RTT and it allows the RTT for all pack-

ets to be calculated in the presence of the delayed ACK option, although as discussed in [JBB92], the sender must be less aggressive in using the RTT for retransmission time out (RTO) calculation.

In order to examine the behaviors of the delayed ACK option on different platforms, we conducted experiments on both Windows and Linux. The Linux kernel versions we tested are 2.4.21 and 2.6.11, while for Windows we used Windows 2000 and Windows XP. Our experiments show the two Linux versions have similar behavior and the two Windows variants behave similarly. However, Windows and Linux do show differences on how ACKs are generated. While both platforms set the maximal ACK delay as 200ms, Linux uses the measured RTT as the waiting time whereas Windows uses a round-robin scheme rotating from 100ms to 200ms. In addition, Linux ACKs immediately during the slow-start phase while Windows tries to acknowledge two data packets if possible even at slow-start. Furthermore, Linux sends an ACK immediately for data packets whose sizes are smaller than 500 bytes assuming they belong to an interactive session. Windows shows no discrimination based on packet sizes. In general, Linux reacts more aggressively than Windows on ACK generation.

We conducted experiments on the three links described in Section 6.2 for both directions. We varied the client receiver platform between Linux and Windows. During the tests, packets were captured at both endpoints using tcpdump with millisecond time granularity. For comparison, [VLL05] recently found about 75% of popular Web servers support the existing TCP Timestamps Option with the majority using a 10ms timestamp granularity. One problem is that the timestamps given by tcpdump are at the data

link layer when packets are just received from or to be sent to the network card, while timestamps in the TCP option are given by the transport layer. The two sets of timestamps may have discrepancies due to the internal processing and queuing time when packets are passed between the two layers. However, we compare the timestamps included in the current TCP Timestamp Option with the corresponding timestamps marked by tcpdump in our experiments, the results do not show any difference if using the lower time granularity of the two sets.

We used the "2Con" approach described in Section 6.2 to generate FTP-like traffic simultaneously in both directions over two separate connections. Again the file size is set to 1MB on both sides. We use two-way traffic to introduce cross traffic in the reverse direction, which may influence the latency of ACK packets. As expected, we only found reverse traffic to have an effect for the DSL client connected with asymmetric bandwidth. We did run each test with and without the delayed ACK option, but all results shown use the default where the delayed ACK option is turned on. We also introduce tests with the download of Web and streaming data from actual Web sites as described in the following section.

7.3 Results

Using the file transfer test with 1MB of data, Figure 7.4 shows one of the more pronounced cases for differences of using ACK spacing to calculate spacing of received data packets. The results are for data sent from California to a client at WPI running on a Windows platform. The results in the

CDF are obtained by comparing the data packet reception spacing (drecv) with the ACK send spacing at the receiver (asend) and ACK receive spacing at the sender (arecv). The results show the cumulative differences between the data spacing and each of the ACK spacings. In computing the difference in spacing between data and ACKs, if the sending or receiving of a delayed ACK represents the reception of two data packets, half of the ACK spacing is used to approximate each of the data spacings.

The significance of these results is that for over 50% of the packets there is a difference between the actual data reception spacing and the approximated spacings obtained using the ACK sent or ACK received results. In the figure, the two data lines overlap for these results because there is no congestion in the reverse direction. The average absolute difference is 60ms as shown in Table 7.2, which contains a summary of all file transfer tests. There are two important points about the calculation of results in Table 7.2: 1) we count a difference or delay as zero if it is less than 1ms; and 2) we use absolute values of packet spacing differences to calculate mean and median values.

Table 7.2 shows the spacing differences are almost non-existent for the California to WPI connection when the client receiver is running Linux. The reason for the different results for a Windows client versus a Linux one is because there are many packet losses (about 10%) on the path from California to WPI for this test. Linux responds with an ACK immediately after a packet loss is detected and continues to respond with one ACK for each received data packet for a while even after the lost packet is recovered, which causes almost one ACK for every data packet for a high loss path.

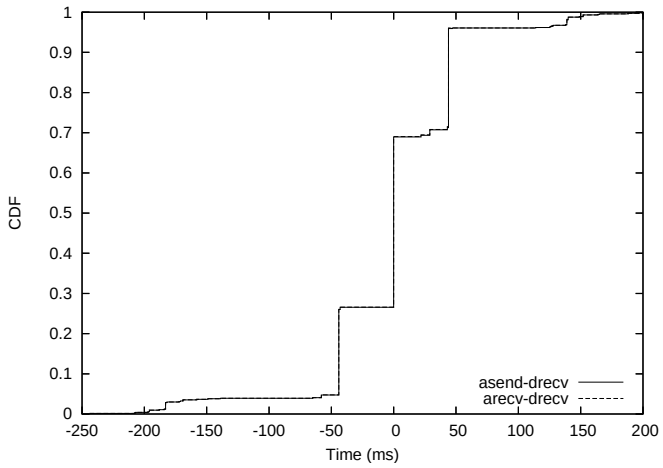


Figure 7.4: Packet Spacing Difference among Data Recved, ACK Sent, and ACK Recved for California to WPI Windows Client

Table 7.2: Summary of Packet Spacing and ACK Delay Under File Transfer Traffic (Times in ms)

Connection	Recv O.S.	|ACKSnd-DataRcv| > 0			|ACKRcv-DataRcv| > 0			Last ACK Delay > 0			First ACK Delay > 0		
		%	Mean	Med.	%	Mean	Med.	%	Mean	Med.	%	Mean	Med.
WPI→IT	Linux	8.8	26	19	13.9	17	7	2.1	41	40	56.4	2	1
IT→WPI	Linux	38.9	16	7	40.2	16	6	3.2	44	41	23.8	7	1
	Win	19.6	17	3	20.1	16	3	0.3	101	101	10.3	15	1
WPI→Calif	Linux	38.7	17	10	42.9	15	8	3.5	34	32	23.7	19	26
Calif→WPI	Linux	0.5	25	10	3.1	5	1	0.3	25	10	0	0	0
	Win	57.6	60	44	57.5	60	44	5.9	146	139	46.0	87	88
DSL→WPI	Linux	28.6	4	1	36.2	15	4	0	0	0	100.0	30	30
WPI→DSL	Linux	13.6	150	3	92.9	33	7	0	0	0	100.0	8	8
	Win	20.5	73	87	99.9	21	7	0.3	110	110	100.0	8	8

Windows also sends an ACK immediately once packet loss is detected, but right after the loss is recovered, delayed ACK processing is immediately restored.

Except for these two extreme cases, the other file transfer tests in Table 7.2 show non-zero differences between the data reception spacing and ACK sending spacing for 8-38% of the packets received with an average difference generally between 15 and 25ms.

When using ACK reception spacing to infer data reception spacing, the estimation error is further enlarged as the reverse side congestion causes ACKs themselves to get delayed. For the path from WPI to the local DSL host, ACKs are transmitted with varied latency due to the congestion in the reverse direction. The ACK delay variance causes a difference for each estimation that is on average 20-30ms in magnitude.

For the same California to WPI Windows client connection used for results in Figure 7.4, the "Last ACK Delay" line in Figure 7.5 shows the delay between when an ACK is sent and the last data packet was received. As shown, only about 6% of ACKs show a non-zero delay. This result is not surprising given that under FTP-like traffic, the receiver generally receives data as quickly as the sender is allowed to send it and more or less immediately generates the corresponding ACK, although occasionally the ACK is delayed for over a 100ms. In the case of a generated ACK covering the receipt of two packets, the "First ACK Delay" results in Figure 7.5 show that there is no delay between the ACK and this first packet more than half the time, but for approximately 45% of these ACKs the first of the two packets had arrived 85-90ms ago. This "First ACK Delay" is not so much an issue

of the receiver's TCP implementation, but reflects differences in the spacing between reception of the first and second data packet. Complete ACK Delay results for all tests are shown in Table 7.2. The results in the table show little delay to generate an ACK after the last packet is received, but frequently there is a delay between the receipt of the first and last packet of a pair.

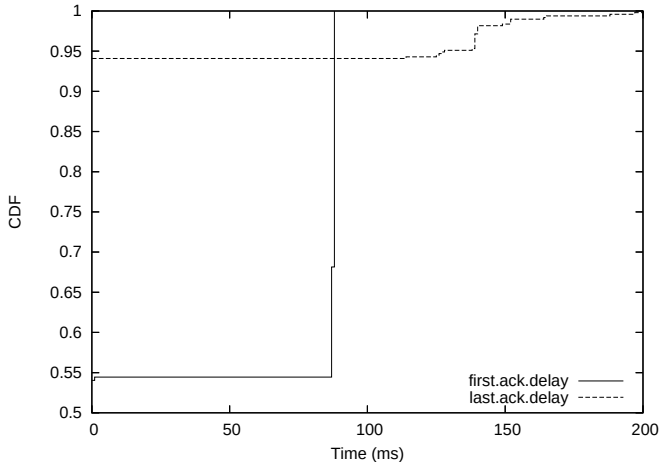


Figure 7.5: CDF of ACK Delayed Time for First and Second Data Packets for California to WPI Windows Client

With FTP-like traffic, we have established the usefulness of including explicit delay information in ACK packets to help the sender more accurately determine data reception spacing, which can be used to infer forward jitter. We postulate the method is more useful for Web and streaming traffic, where Web objects or streaming frames are not always sent at the rate allowed by TCP as is typical with during a file transfer. ACK delay on the receiver side is more significant. Table 7.3 shows the results for ac-

Table 7.3: Summary of Packet Spacing and ACK Delay Under Web and Streaming Traffic (Times in ms)

Traffic Type	Connection	Recv O.S.	|ACKSnd-DataRcv| > 0			Last ACK Delay > 0			First ACK Delay > 0		
			%	Mean	Med.	%	Mean	Med.	%	Mean	Med.
Web	CNN→DSL	Win	81.1	853	20	52.4	50	4	100.0	13	7
	CNN→WPI	Win	87.0	427	26	69.0	66	57	7.7	10	10
		Linux	54.2	814	15	17.1	25	39	12.5	29	29
	Cisco→DSL	Win	70.9	346	48	33.8	84	101	95.3	18	8
	Cisco→WPI	Win	73.6	155	44	34.1	81	107	41.4	37	2
		Linux	56.5	93	40	21.2	40	40	32.3	2	2
Streaming Audio	Amazon→DSL	Win	90.3	7	2	100.0	111	110	0	0	0
	Amazon→WPI	Win	30.9	3	1	100.0	108	107	0	0	0
Streaming Video	Yahoo→DSL	Win	12.3	9382	121	0.3	111	130	99.6	8	8
	Yahoo→WPI	Win	24.0	2319	7	1.1	143	151	12.0	12	13

cessing two Web site home pages and two streaming media sites from two different client locations. We used the Internet Explorer (IE) browser for Web access on the Windows platform and Firefox for access on the Linux platform. Each browser used persistent connections. We were unable to access the streaming media sites using a Linux client. For the Amazon audio stream, which provides online listening for sample music, we used Real Player, a plug-in to IE. For the Yahoo video, which provides movie previews, we used the Window Media Player, also a plug-in to IE. As we only have client side packet traces, we are not able to determine the difference between ACK reception spacing and data reception spacing.

The "Last ACK Delay" and "First ACK Delay" results in Table 7.3 show significantly higher non-zero percentages as well as mean and median values than those results under FTP-like traffic in Table 7.2. In particular, Figure 7.6 shows the CDF of ACK delays when accessing the CNN Web home

page from a Windows platform. Around 70% of ACKs are delayed from receipt of the last data packet and the delay varies from several milliseconds to close 200ms. For the cases when one ACK acknowledges two data packets, both "First ACK Delay" and "Last ACK Delay" are small, but this situation only occurs for around 30% of the total packet counts.

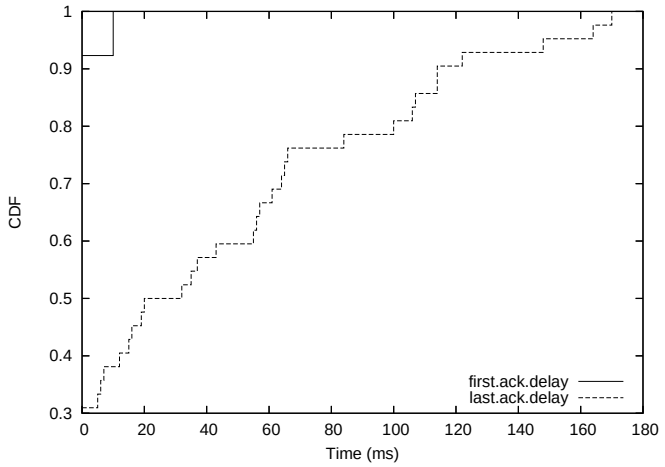


Figure 7.6: CDF of ACK Delay for CNN Web to WPI Windows Platform

Another interesting result is the ACK delay for the Amazon streaming audio to the Home DSL Windows platform. The last ACK delay is nearly a constant 110ms and there is no first ACK delay because each ACK is generated for exactly one data packet and the receiver is trying to wait for a second packet to arrive. The constant ACK generation delay does result in little difference between the data reception and ACK generation spacing for the Amazon to Home DSL connection as shown in Table 7.3. However, in general there is a large discrepancy between these two spacing calculations in Table 7.3 because the sender of Web and streaming traffic is not

always sending packets and the receiver delays waiting for a second packet to arrive before generating the ACK. These gaps between when data packets are sent combined with the delayed ACK feature mean that the gap between ACK transmissions is an unreliable estimator for the gap between data packet arrivals.

The final benefit of having more complete data and ACK packet transmission information is to calculate more accurate path RTT than using the Timestamps Option. The difference is trivial when there is no ACK delay and packet loss. However, in many cases as shown in Table 7.2 and 7.3, ACKs are delayed and consequently the RTT is overestimated. The situation becomes much worse when packet loss occurs because the RTT calculation could include packet retransmission time [JBB92].

As an example, results in Figure 7.7 show a file transfer from California to WPI where the path involves around a 10% packet loss. The RTT calculation based on RFC 1323[JBB92] is 2-4 times higher than the real RTT. The sender may treat the enlarged RTT as a sign of congestion and react conservatively. However, in some cases the RTT and packet loss should be decoupled such as in a path where routers use Random Early Detection (RED) [FJ93] rather than drop-tail as the algorithm to handle congestion, or in a path where a wireless segment is involved. Having a much larger estimated RTT compared to the real RTT could cause retransmissions to occur much later than they should.

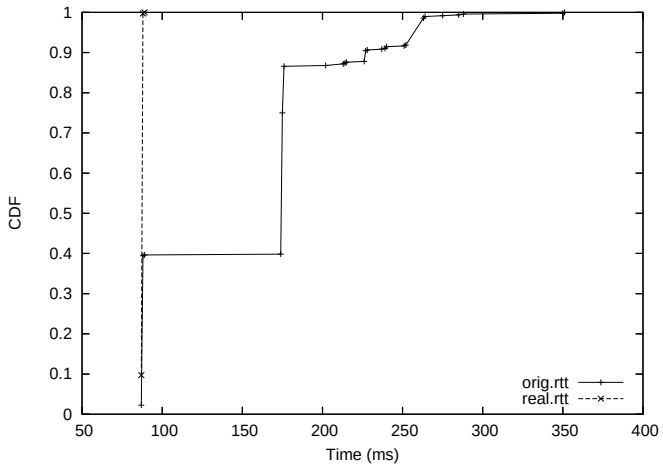


Figure 7.7: Comparison between RTT Based on the Original Timestamps Option and RTT Calculated Based on Enhanced Timestamps Option: California to WPI Link

7.4 Observations

We obtained many useful results from the enhanced ACK experiments to maintain more complete packet transmission information at the receiver in a TCP connection and share this information with the sender via a new Timestamps Option. A desirable feature in a TCP connection is to be able to know the spacing between data packets received at the receiver and hence establish jitter in the forward data transmission channel. Currently this spacing can only be inferred from spacing between ACK generation at the receiver or ACK reception at the sender. However, results from many file transfer experiments show discrepancies between the spacing of ACKs and the actual spacing of received data packets. In one file transfer experiment we found a difference between these spacings in over 50% of cases with

an average difference of 60ms. These differences are due to reverse channel congestion, variation when data packets are received, which is masked with the delayed ACK feature, as well as delays when ACK packets are generated by the receiver. These differences occur more frequently with generally a larger magnitude when traffic is not sent as frequently as allowed by TCP as can be the case with Web or streaming data. All of these variations, combined with packet loss, make the determination of the actual RTT difficult for the sender resulting in conservative estimates to be made in determining the RTO for the connection.

The knowledge of the data packet inter-arrival time at the receiver side may help congestion control schemes that are based on bandwidth estimation such as TCP Westwood [MCG+01]. TCP Westwood uses the inter-arrival spacing of ACK packets at the sender side to estimate available bandwidth in the forward direction. The bandwidth estimation is complicated when delayed ACK or ACK compression [Mog92] in the reverse direction occurs [GM02a]. These problems can be avoided if the inter-arrival information of data packets at the receiver side is explicitly available to the sender.

If delay jitter information is available to applications (e.g. by extending the "tcp_info" structure [tcpb]), applications can have a better understanding of current network conditions. For example, the delay jitter for data packets sent back-to-back roughly reflects the bandwidth of the bottleneck link of the path. The steady increasing of delay jitters can be used as a sign for congestion on the way. With the enhanced TCP timestamp option, delay jitter can be measured for each direction and the congestion in the

reverse direction will not influence the measurement in the forward direction. Applications such as streaming may use that information to tune the data sending rate.

Traditional TCP implementations including TCP NewReno [FH99] use packet loss as signs for congestion on the path and reduce congestion window correspondingly. This conjecture does not work well for paths including wireless segments, where packet loss is mainly caused by poor wireless signal [GM04]. A TCP implementation can adopt the enhanced timestamp option to help differentiate the congestion from loss for wireless. A simple method is to use RTT variance along with packet loss to infer network congestion other than using packet loss alone. This method is not possible with the current RTT calculation using the traditional timestamp option, which already includes the factor of packet loss besides ACK delays. Taking the path condition shown in Figure 7.7 as an example, the high loss rate but constant RTTs together imply that a loss-prone link is more likely involved rather than congestion on the path. The RTT measurement using the traditional timestamp option (also shown in the Figure 7.7) shows a big variance of RTTs and cannot be used to distinguish these two situations.

An important point concerning the availability of this more complete and accurate information about the TCP connection is that it is obtained with minimal, bounded overhead by the TCP receiver and only a small amount of additional information (12 bytes more in the case of accumulated ACKs and 4 bytes more in the case of non-accumulated ACKs) added to an ACK packet that must be sent anyways. Results from our data piggybacking experiments show that adding a small number of bytes to reverse-

channel traffic over even a bandwidth-constrained link can be accommodated.

If dropped or out-of-order packets occur with the enhanced ACK approach as did occur in some of our experiments then its additional information provides a more complete picture of what is happening with the data transmission, although the approach would complement, not replace, the existing selective ACK option [MMFR96] in retransmission of missing packets.

An interesting question is whether the data piggybacking approach and the enhanced ACK approach should be combined. Because the enhanced Timestamps approach is proposed as a TCP option it could be combined with the data piggybacking approach. In terms of whether the combination makes sense, the enhanced ACK information could help both endpoints of a connection determine the available bandwidth between them, although use of the proposed option in the direction of data flow would cause a small reduction in the per-packet data capacity for that direction. Further research is needed to study the relative tradeoffs of combining the approaches.

7.5 Summary

In this work we have discussed and evaluated the enhanced ACK approach, which is an extention to the existing TCP Timestamps Option. The enhanced ACK option allows a TCP sender to track the spacing between all data packets arriving at the receiver and to have more complete timing

information for the forward and reverse directions of the connection. The information can be used to better detect jitter and congestion in the forward direction than what is currently available.

Results from many file transfer experiments show discrepancies between the spacing of ACKs and the actual spacing of received data packets. These differences occur more frequently with generally a larger magnitude when traffic is not sent as frequently as allowed by TCP as can be the case with Web or streaming data. The new option also allows a more accurate RTT estimation, which is not influenced by the factors of ACK delay or packet loss. Our results show that the current mechanism of RTT calculation can make over 200ms overestimation when packet loss occurs.

Chapter 8

Packet Aggregation

Previous results in Section 3.3 indicate that less than 20% of packets reach the full MTU size and many packets are small. The median and mean sizes for all IP packets are about 200 and 550 bytes respectively. These results are consistent with other work [TMW97, MC00, FKMkc03, FML+03] conducted at different time periods and locations. At the same time, we have also observed numerous concurrent flows existing between the same host or cluster pair. These facts bring the opportunities to aggregate small packets into big packets to save the total number packets for transmission. Reducing the total number of packets helps alleviate workload of intermediate routers whose processing cost is packet-based instead of byte-based. By sharing the common IP headers, a certain amount of overhead can be reduced, therefore improving the bandwidth usage efficiency.

Previous work has addressed aggregation at different levels. Approaches like HTTP/1.1 [FGM+99]), SCP [Spe], and SMUX [GN98]) are at the application or session layer. They all seek to multiplex independent data streams

over one TCP connection in order to reduce the cost of opening and closing TCP connections. As a side effect, they may result in less packets because of more data to be sent on one connection.

The Nagle Algorithm [Nag84] used in TCP and the built-in multiplex option in SCTP [SXM+00], both at the transport layer, allow user data to be aggregated into packets. TCP uses Nagle's algorithm to force user data to be accumulated before all acknowledgments are received or a full MTU size packet is filled up. SCTP allows bundling of multiple user messages into one SCTP packet. The aggregation efforts for both TCP and SCTP are limited to one TCP connection or one SCTP association.

The "car pooling" method [BS99] aggregates packets at the network level. This work suggests placing aggregators and splitters in desirable locations of the network. Aggregators combine small packets going to the same destination while splitters regenerate the packets at the destination. Under a simulated environment, "car pooling" shows performance gain when assuming the bottleneck is the number of packets that can be queued.

Packet aggregation is based on the same idea as the "car pooling" method. However, we evaluate the performance of packet aggregation in a different way. In the "car pooling" work, network traffic is simulated as 100% Web request-response traffic, where the object size is fixed and the packet size is a uniform 536 bytes. In addition, the aggregation efforts are based on a hard limit of how many packets can be aggregated instead of using MTU as the boundary. Furthermore, the queue size of routers, represented as the number of packets, is assumed as the path bottleneck in spite of other factors like bandwidth and buffer size in bytes. The throughput gain shown

in the results largely depends on the above assumptions, which may not be realistic or representative.

We adjust the evaluation method in the following aspects. First, we use real traffic traces as the input to a packet aggregator instead of simulated Web traffic. Second, we use both MTU size and a deadline as constraints for aggregation. We use the Ethernet MTU as the space boundary for aggregated packet size and a deadline as the time boundary for how long a packet can be held before it must be sent out. Third, we use the reduced number of packets instead of improved throughput as the metric to evaluate the gain of packet aggregation. Throughput is an intricate measurement that depends on many factors such as topology, bandwidth, traffic dynamic, and router queue management disciplines. We pick the number of packets as a simple indicator for upper-bound improvement by using packet aggregation.

8.1 Measurement Method

We use a trace-based simulation to evaluate the packet saving benefit that can be achieved by packet aggregation. The trace logs are described in Table 3.1 of Chapter 3. For the ISP logs, each record is a packet trace, which can be used as input to the simulation directly. However, for the WPI logs, each record is a summary of the number of packets and total bytes in the last minute of a flow. We use the following method to estimate packet sizes in a record. 1) For a record that has SYN or/and FIN flags, we consider this record to have one/two packets that have the minimal TCP packet size

of 40 bytes and exclude the packets from the following calculation. 2) For TCP, given the number of packets and bytes in a record, we calculate the maximal number of packets that can carry the full MTU size of 1500 bytes. We then count all the remaining packets as non-full packets and they have the same packet size. 3) For UDP, we simply use the average size for all packets in that record. We find that this method gives a good estimation on packet sizes. We compare the packet size distributions for particular types of flows between the ISP logs and WPI logs, and the results are very close.

For WPI logs, we also assume that the packets in a record are evenly distributed for the duration of the record. When two flows have a overlap, the available space of one flow in the overlap can be used by the other. This method simplifies the simulation but may overestimate the aggregation effects by giving more flexibility of how the available space can be used. For this reason, we only use the results for the WPI logs as a upper bound for the benefit that can be gained by packet aggregation.

We examine aggregation effects on two types of scopes: host-to-host and cluster-to-cluster. For the host-to-host scope, we assume that both the aggregator and splitter exist at the end hosts. For example, we can add the aggregation/split function to the IP layer of TCP/IP kernel. All packets are subjected to be aggregated before being sent to the data link layer and all aggregated packets are split once the network layer received them from the data link layer. For the cluster-to-cluster scope, we assume that the combination of aggregator and splitter exists for each cluster that we have identified with BGP routes (Chapter 3). The aggregator/splitter can be a separate unit sitting in front of the edge router for the cluster or be

combined with the edge router itself. All traffic from the cluster is subjected to aggregation. We use the same clustering method as described in Section 3.1. For WPI logs, we use BGP routes to classify IP addresses, while for ISP logs we just use traditional class C.

For the purpose of aggregation, small packets are delayed in the hope that they can combined with other small packets. We set a *deadline* for how long a packet can be held before it has to be sent. Ideally, an application can set the deadline value for each message it generates because a deadline can be different from application to application or from message to message. However, in most cases, a deadline can be set for each type of application. For example, interactive applications are given a short deadline and file transfer applications can have a longer deadline. In the following experiments, we apply a fixed deadline for all applications while varying the deadline value for each experiment. We use such a simplified prototype to quickly understand the benefits of aggregation and the influence of different deadline parameters.

Another constraint for aggregation is the maxim packet size. As all the traffic logs used in the following experiments were collected at an Ethernet subnet. We use the Ethernet MTU size as the upper boundary for the aggregated packet size. It is possible that an actual path MTU [MD90] is smaller than the MTU size we use. We consider these cases are rare as nearly all current networks support MTU size equal to or bigger than 1500 bytes and the MTU size tends to be larger as proposed in [KSO+01]. We calculate the size of an aggregated packet as equal to the sum of the sizes of its all component packets, including IP and transport layer headers. Delivery of

such aggregated packets can be done through encapsulation of multiple IP packets into a single IP packet as is currently done for tunneling [Per96].

Aggregation may cause packet reordering problems, where a packet sent earlier is received later than another packet due to aggregation delay. Packet reordering is harmful if the reordered packets belong to the same flow. We avoid packet reordering by enforcing packets of the same flow be sent in order. When a packet must be sent out (e.g. a full-size packet), all previously delayed packets of the same flow must be sent out immediately before this packet.

8.2 Results

We first perform aggregation on all packets without regarding whether these packets are from the same flow or different flows. We empirically pick 0.05 and 1 second as two fixed deadline values for each experiment under different scopes. For most applications, 50ms is a tolerable delay and we push the deadline to 1 second to examine the improved aggregation effects when using a much longer delay. The results are shown in Table 8.1.

For the two ISP logs, there is about a 20% packet reduction when using 0.05 second deadline threshold and this number increases to nearly 50% when using 1 second deadline. The results under the cluster-to-cluster scope do not show much bigger improvement than these under the host-to-host scope. Results for the three WPI logs show higher percentages of packet reduction than the two ISP logs. One reason is that the WPI logs have little P2P traffic as P2P applications are forbidden due to campus pol-

Table 8.1: Percentages of Packet Reduction by Aggregating All Packets

LogName	Host-to-Host		Cluster-to-Cluster	
	0.05Sec	1Sec	0.05Sec	1Sec
wpi1	28.3%	55.8%	29.5%	56.3%
wpi2	29.1%	56.1%	29.7%	57.0%
wpi3	29.3%	56.2%	30.1%	57.3%
isp1	18.5%	44.7%	18.8%	45.0%
isp2	19.3%	47.5%	20.3%	48.5%

icy. On the other hand, about half of the packets in the ISP logs belong to P2P applications. P2P traffic generally has large packet size and small aggregation opportunities. Another reason is that the method used to estimate packet traces from a flow record for the WPI logs exaggerates aggregation effects to a certain extent.

The results look promising as a significant amount of packets can be saved. However, a closer check reveals that most aggregation takes place between packets inside a flow, especially the ACK packets on the reverse direction of a TCP flow. Packet aggregation inside a flow has several problems. First, aggregating ACKs on the reverse direction causes the ACK compression problem [BPS+98], which is undesirable as TCP uses reception of ACKs to pace the transmission rate. Second, a dependency exists between packets within a flow. If the generation of a succeeding packet depends on the response of a previous packet, these two packets cannot be aggregated. Third, packet aggregation within a flow can be done and most appropriately done at the application level, which obeys the ALF (Application-Level Framing) principle in [CT90] that suggests applications

should take control of data sending units.

As aggregation is undesirable for packets within a flow, we limit aggregation to take place only for packets belonging to different flows (henceforth called *inter-flow packet aggregation*). The results are shown in Table 8.2.

Table 8.2: Percentages of Packet Reduction by Aggregating Only Inter-flow Packets

LogName	Host-to-Host		Cluster-to-Cluster	
	0.05Sec	1Sec	0.05Sec	1Sec
wpi1	8.9%	9.8%	18.5%	19.6%
wpi2	9.5%	10.5%	19.3%	20.4%
wpi3	10.0%	10.9%	20.1%	21.1%
isp1	2.9%	3.8%	3.9%	5.9%
isp2	2.8%	4.0%	4.1%	6.3%

Not surprisingly, the number of packets that have been reduced is much smaller than that when aggregation is applied on all packets. For the two ISP logs, there are only 3-6% packet reduction for all experiments. These small reduction percentages are largely because of the massive P2P traffic in ISP logs. Using traditional class C to cluster IP addresses also underestimates the aggregation effects under the cluster-to-cluster scope. The results for WPI logs show higher packet reduction, about 10% under the host-to-host scope and almost doubled for the cluster-to-cluster scope. Using a longer deadline does not show a significantly higher gain than using a short deadline.

Despite the marginal savings on the total number of packets, packet aggregation introduces additional delay as well as requires more state in-

formation such as deadlines for packet transmission. In addition, less and fatter packets are not always good. When packets are transmitted over wireless, bigger packets have higher chance of getting errors. The optimal packet size to reach highest goodput is a function of the current error bit rate. Similarly, intermediate routers using rate-based dropping schemes such as adaptive virtual queue [KS04] also have a higher possibility to drop bigger packets. Finding optimal transmission size for this type of scheme is out of the scope of this study. However, assuming that size is known, packet aggregation can treat it as an effective MTU for the path and only aggregate packets up to that limit.

8.3 Summary

While aggregation over all packets has a significant packet savings, the results for inter-flow aggregation show much less of a gain. Despite that the "car pooling" work [BS00] presents much performance improvement by using packet aggregation, our results show only marginal packet savings in more realistic experiments. For traffic involving many P2P flows, packet aggregation is not effective. We observed only about 6% packet reduction even under the cluster-to-cluster scope. For traffic exempting P2P flows, the gain is higher, but still not much. We show the upper bound of packet savings for two such logs are about 10% under the host-to-host scope and 20% under the cluster-to-cluster scope. On the other hand, aggregation introduces extra delay and more implementation complexity to TCP/IP stacks. Comparing the cost and gain, we do not see this direction

as appropriate for further investigation unless traffic patterns change in the future.

In a closer examination, we find that the low aggregation ratio is caused by a small number of flows that involve large file transfers. This amount of flows takes less than 10% of total flows but account for over 90% of total packets. The observation is called the "elephants and mice" phenomenon in previous studies [SRS99, FP99, BDJ01, SRB01, PTB+02, cLH03a]. Elephants, i.e. flows having big transmission size, only take a small share of flow counts but contribute the majority of network traffic. Most packets in the download direction of these flows are full and aggregation is fruitless.

Chapter 9

Critical Packet Piggybacking

Previous results for packet aggregation in Chapter 8 did not show much packet savings. While aggregation is ineffective for elephant flows, an interesting question is whether available packet space should and could be used to help other flows. In one study [EV01], the authors point out that routers should focus on the elephants and ignore the mice. It may be important for routers as the elephants take most of link bandwidth. However, the statement is not true from the end user point of view. The elephants, which usually involve large file transfers, do not have as critical performance requirements as interactive or transactional applications. A P2P user normally puts the downloading process in background or leaves it unattended overnight. In contrast, an interactive or transactional session requires the attention of an end user all the time. These types of sessions normally generate mice flows and their performance is critical to user-perceived latency. In terms of flow counts, mice flows take the majority share of total flows and are our focus in this chapter.

Interactive or transactional sessions are sensitive to round trip time and packet loss. In these types of sessions, the generation of the next message is normally based on the response to the previous message. The request-response delay highly depends on the round trip time between the two end hosts. When packet loss occurs, the request-response delay may become much longer due to timeouts and retransmissions. For example, the time-out for resending a DNS request is 3 to 5 seconds depending on different DNS client implementations. The loss of TCP SYN or SYN ACK causes a timeout of 3 seconds by default on both Windows and Linux systems. Even after a TCP connection has been established, the minimal retransmission timeout (RTO) value is still 200 milliseconds in the Linux operating system [SK02b] and one second in many other systems as suggested by [AP99]. The fast retransmission and recovery scheme [Ste97] is not likely to take control in interactive and transactional sessions as there are not enough packets to send in one RTT.

To protect packets from loss and incurred timeout, we propose a scheme called *critical packet piggybacking*, which protects critical packets from loss by exploiting available packet space in other concurrent flows. We define *critical packets* as the packets that are significant in the critical path for the performance of an application. For example, the handshake packets for a TCP connection establishment are critical for applications as the loss of SYN or SYN ACK packets will incur a timeout measured in seconds. DNS query and response messages are also critical as the loss of either message will add a 3-5 second delay. Another example is the I frames in MPEG video streams [MPFL96]. Loss of one I frame causes many unusable P and

B frames that depend on it. To protect these critical packets, we can either send duplicate copies or redundant forward error correction (FEC) data for them. With the available space provided by other flows, these data can be piggybacked without introducing extra packets.

Critical packet piggybacking can also be used for predicted packets. Predicted packets shortcut the critical path if the prediction is accurate. But it introduces unnecessary traffic if the prediction is wrong. Previously in Chapter 5, we sent predicted responses with the first legitimate response without adding any extra packets. With available space provided by other flows, more piggybacking opportunities exist.

Although both critical packet piggybacking and packet aggregation seek to use the available packet space provided in other concurrent flows, they have different objectives. Packet aggregation applies to all non-full packets and seeks to reduce the total number of packets. Critical packet piggybacking only applies to packets that are critical for application performance and protects them from packet loss. Elephant flows have a big influence on packet aggregation, but little impact on critical packet piggybacking as there are not many critical packets in these flows. Critical packet piggybacking does not necessarily require a lot of available packet space. For the critical packets in interactive and transactional sessions, they are normally small and easy to be piggybacked by other flows.

In terms of costs, while critical packet piggybacking does not introduce additional packets, it adds more bytes. Precautions should be made when deciding which packets should be protected from loss. Small but performance-critical packets are good candidates when considering the ra-

tio of value over cost.

One issue with sending duplicate packets is that these packets must be idempotent, where reception of duplicate copies of the same packet should have the same effect as receiving exactly one such a packet. This problem can be resolved by setting a tag for the duplicates or just sending FEC codes instead of the original packets. Upon receiving a duplicated packet, the receiver should silently discard it if it already gets the original one.

Another issue with critical packet piggybacking as well as packet aggregation discussed in Chapter 8 is that they introduce delay in favor of aggregation. This delay influences RTT calculations and cannot be excluded even with the help of the enhanced TCP timestamp option we proposed in Chapter 7. The reason is that aggregation takes place at the network layer and TCP timestamps are marked earlier at the transport layer. TCP does not know whether a packet will be delayed in the future. One solution to this problem is to let the network layer remark the sending timestamp in TCP timestamp options for delayed packets. If a delayed packet is a data packet, the remarked timestamp reflects when this packet leaves the machine and will be echoed back in its corresponding ACK. The aggregation delay therefore will not be included in RTT calculations. If a delayed packet is an ACK packet, the remarked timestamp reflects when this ACK leaves the machine. With the help of the enhanced timestamp option, this aggregation delay is counted as part of the ACK delay, which is excluded in RTT calculation as well.

Results in Chapter 3 show the general existence of flow relationships and the presence of available packet space. In the following, we focus

on several scenarios where critical packets can be piggybacked by related flows. We examine how much available space can be provided by these related flows and how many critical packets can be piggybacked. These scenarios are:

1). Packet piggybacking for an interactive application. We choose SSH (Secure SHell) as a representative interactive application, which involves interactions between an end user and a remote service. Due to its secure feature, SSH is much more common in our traffic logs than its predecessors like "telnet" or "rlogin". Packets inside a SSH session normally enclose commands and responses between the user and the server. They are generally small and can be easily piggybacked by other flows. The SSH application is vulnerable to long RTTs and packet loss. The loss of one packet normally incurs a timeout and packet retransmission happens thereafter. As a way to recover quickly from a packet loss, the SSH application can ask the transport layer to send certain packets in duplicate if piggybacking is possible.

2). Packet piggybacking for a Web application. The Web provides a versatile interface for multiple purposes. On one hand, the Web flows involving large file transfers are generally large in transmission size. On the other hand, flows for Web transactions are normally small in transmission size. As packet piggybacking is ineffective for the former case, we study piggybacking opportunities only for small Web flows. As with the SSH application, small Web flows are also vul-

nerable to long RTTs and packet loss. Sending duplicate packets for certain critical data (e.g. Web object requests) helps protect against packet loss. Furthermore, due to relationships between Web objects, it is possible to predict and piggyback future packets.

3). Packet piggybacking for streaming applications. We pick "real" (realplayer streaming application) as it is one of the most popular streaming media applications. Packets inside a streaming flow have different significance. Packets that are more critical than others can be protected by sending redundant data. These redundant data can be piggybacked with other concurrent flows without introducing extra packets.

4). Packet piggybacking for TCP establishment. A TCP establishment procedure includes a three-step handshake. Any packet loss during this procedure will incur a long timeout. We can avoid this delay by letting other concurrent flows piggyback protective duplicates for these packets. In addition, if we can foreshadow upcoming future flows, the connection establishment procedure(s) can be piggybacked by ongoing flows, as a shortcut in an application's critical path.

9.1 Measurement Method

We use a similar method as we did to examine flow relationships in Chapter 3. However, in addition to examining temporal overlaps between flows, we also count the number of non-full packets inside a flow and check to

what extent these non-full packets can be piggybacked by concurrent flows. For the ISP logs, we have packet size information for each packet. For the WPI logs, we have aggregated information at a one-minute granularity for each flow. We use the same method described in Section 8.1 to estimate the number of non-full packets and their sizes in a record.

For the same reason as we discussed in Section 8.2, we examine the possibilities of inter-flow packet piggybacking only, i.e. piggybacking duplicate critical packets into other flows. We are aware that applications themselves may send duplicates or redundant data within the same flow such as FEC used in some streaming applications [WCK05, FB02, LC00]. It is purely an application-level decision. Here we try to understand to what extend piggybacking is possible given the existence of concurrent flows.

For the four scenarios described above, we use transmission protocols along with port numbers to identify application flows of interest. This method introduces some inaccuracy when the same protocol and port combination is used for multiple purposes. For example, both SSH and SFTP (Secure FTP) use the same TCP port 22. However, the flow types of our interests are generally much more common in the logs than other flow types that use the same protocol and port number. We believe that the results based on the mixture of flows should not change the tone of observations.

When two flows have a temporal overlap, we assume the available space from the non-full packets of one flow can be used by the other during the overlap period. This assumption exaggerates the piggybacking effects because it grants more flexible temporal constraints for a packet to be piggybacked. We use this estimation because the WPI logs only include

flow traces. Another reason is that under that assumption we only need to maintain states for recent flows instead of for all recent packets, which significantly reduces the spatial and temporal complexity of the simulation.

There are two possibilities for a flow of interest to have temporal overlap with other flows. The flow can have overlaps with flows that already exist when it starts or with flows that begin after its starting time. For both cases, available space in the overlaps can be utilized by the flow in question. However, only for the first case, packets involved in TCP handshakes can be piggybacked. For the latter case, we look ahead for another flow within a one second limit. If a flow begins later than the starting time of the flow in question but within a one second period, we consider it still possible for the following flow to piggyback protective packets for TCP establishment of the first flow. The one second grace period is chosen because if a SYN packet can be resent within one second, it still represents a saving compared with the 3 second default timeout.

9.2 Results

9.2.1 Piggybacking for SSH

As we check the piggyback opportunities for normal SSH flows, we only focus on flows that actually go through the connection establishment stage. For that reason, we filter out TCP flows that have less than 3 packets in either direction. We discard a significant number of flows according to that criteria. The majority of these incomplete flows only show traffic in one direction, which can be explained as connection attempts but no response.

We have examined all the logs included in Table 3.1. The results for all the five logs except wpi3 are similar. However the results for the wpi3 log show a different pattern. A closer examination shows that there are 5 times more SSH flows in the wpi3 log than the other two wpi logs. Among them, over 80% of SSH flows have near constant numbers of packets between 10-13 for each direction and the majority of these flows are initiated from 3 hosts. We suspect that these flows are generated by automatic programs, which skew the results for the wpi3 log.

We show the results for wpi1, which are representative for the four logs excluding wpi3. There are just over 4K SSH flows included in the wpi1 log. Figure 9.1 and 9.2 give the cumulative density functions (CDFs) for the number of packets and the number of non-full packets in each SSH flow in both directions. Here "downstream" means the direction from a SSH server to a SSH client and "upstream" is the reverse direction. About 80% of flows have less than 400 packets in each direction and the median values are around 65 packets. There are about 1% of flows that have over 100K packets in each direction. We conjecture that these SSH flows are actually used for SFTP.

We see that the curve for the number of non-full packets almost overlaps with the curve for the number of packets, which indicates the majority of the SSH packets are not full. The average packet sizes for 90% of SSH flows are under 385 bytes for the downstream direction and under 260 bytes for the upstream direction. The general small sizes of packets suggest that these packets are easy to be piggybacked if there are other concurrent flows.

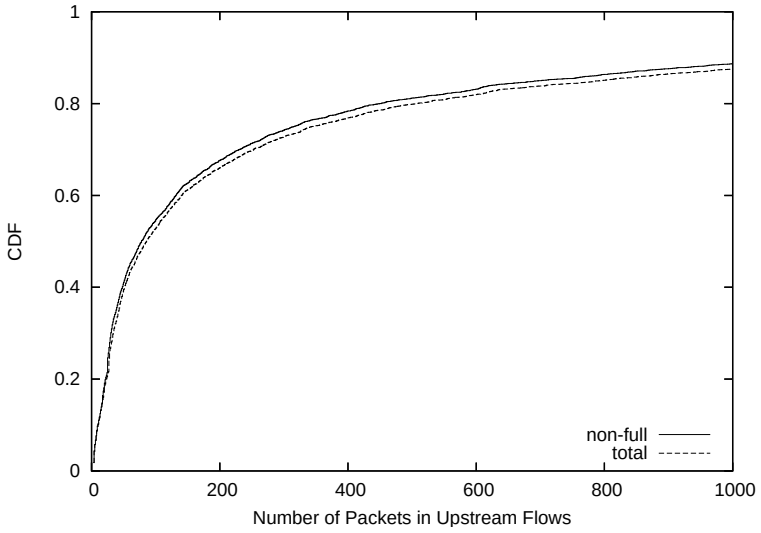


Figure 9.1: CDFs of the Number of Packets and the Number of Non-full Packets in SSH Upstream Flows (for wpi1 log)

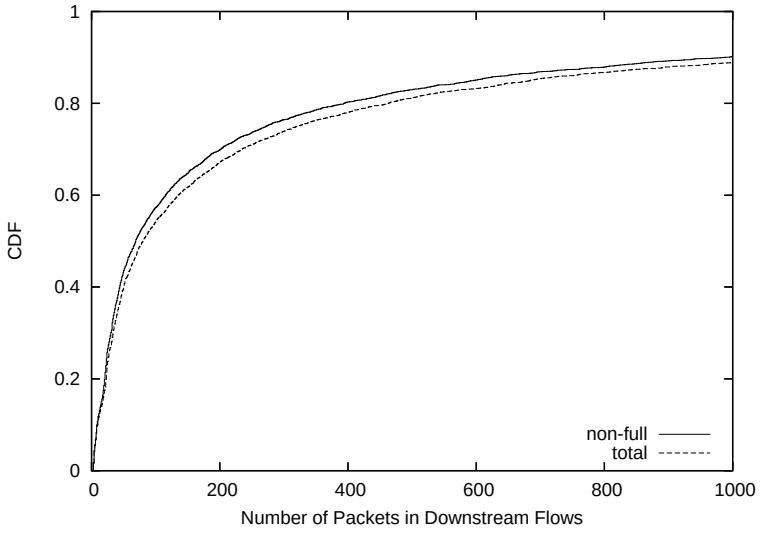


Figure 9.2: CDFs of the Number of Packets and the Number of Non-full Packets in SSH Downstream Flows (for wpi1 log)

We then check the overlaps of SSH flows with other flows. Under the host-to-host scope, there are close to 20% of SSH flows that overlap with other flows. Under the cluster-to-cluster scope, this number goes up to 60%. Most of these concurrent flows are also SSH flows, but for different sessions. Other important concurrent flows are Web and PoP3 flows. We examine how many packets in a SSH flow can be piggybacked given the available space provided by these concurrent flows. The results are shown in Figure 9.3 and 9.4 for upstream and downstream directions respectively.

We calculate the percentage of non-full packets that can be piggybacked over the total number of non-full packets for each flow. The CCDFs (Complementary CDF) for all SSH flows are given in Figure 9.3 and 9.4. We observe that under the host-to-host scope nearly 20% of SSH flows have at least one packet that can be piggybacked by other flows. About 8% of SSH flows can have all non-full packets piggybacked by other flows. Under the cluster-to-cluster scope, the piggybacking opportunities are significantly increased. For 60% of the SSH flows, one or more non-full packets can be piggybacked by other flows within the same cluster pair. Close to 30% of SSH flows can have all non-full packets piggybacked under the cluster-to-cluster scope.

Most flow relationships observed between SSH flows with other flows are caused by an end user opening multiple SSH sessions, browsing Web information, or reading E-mails from the same site. It is more related to user access patterns than inherent application behavior. While we do not observe the existence of a huge amount of relationships between SSH flows with others especially under the host-to-host scope, a considerable amount

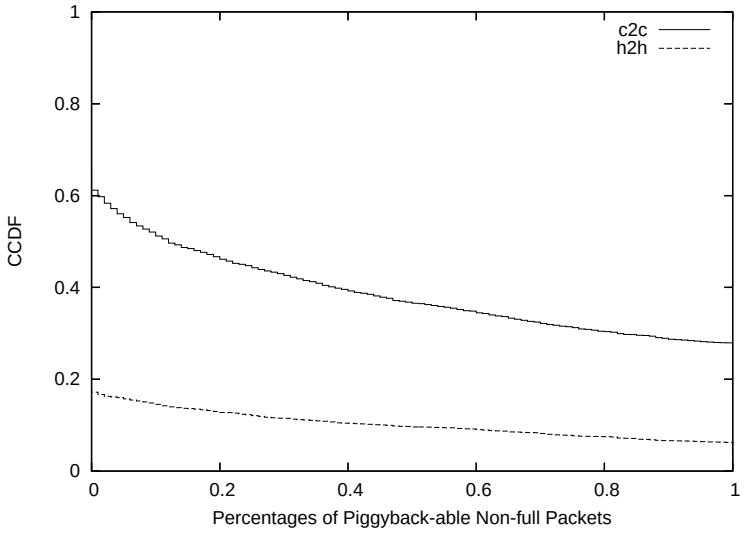


Figure 9.3: CCDFs of Percentages of Piggyback-able Non-full Packets in SSH Upstream Flows under Different Scopes (for wpi1 log)

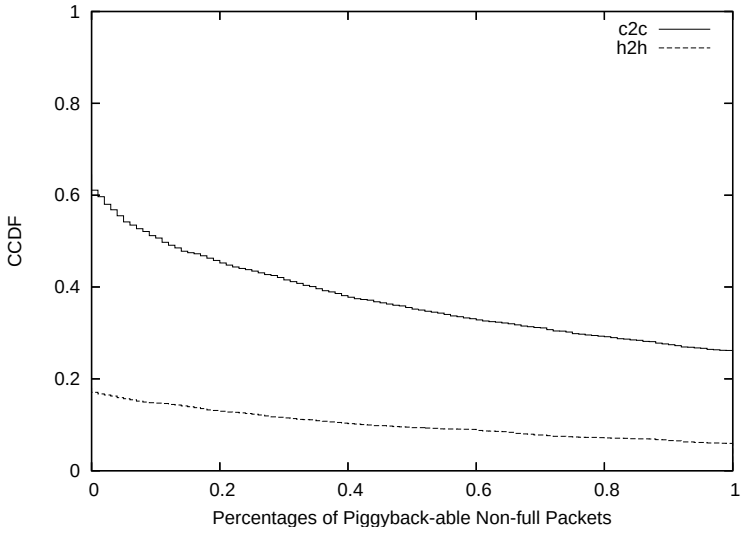


Figure 9.4: CCDFs of Percentages of Piggyback-able Non-full Packets in SSH Downstream Flows under Different Scopes (for wpi1 log)

have overlaps with others. As most packets inside SSH flows are small, there are good piggybacking opportunities whenever overlaps occur. SSH applications can decide which packets should be protected and ask the transport layer to send duplicate copies if piggybacking is possible. One simple scheme is to let the transport layer (i.e. TCP) retransmit every packet that has not been acknowledged in one round trip if piggybacking is possible. This scheme is essentially the aggressive timeout technique we discussed in Section 4.4, but using inter-flow relationships.

9.2.2 Piggybacking for the Web

For the same reason as what we have done with SSH flows, we filter out incomplete Web flows that have less than 3 packets in either direction. For the three WPI logs, Web traffic is dominant for all the TCP traffic. For the two ISP logs, Web traffic is not dominant but still takes a significant share. The results for all five logs are consistent except that the piggybacking percentages under the cluster-to-cluster scopes for ISP logs are lower than these for WPI logs. This difference may be caused by the way we used to cluster IP addresses in ISP logs, which is more conservative than using BGP routes. Again, we only show results for the wpi1 log.

Figures 9.5 and 9.6 show the CDFs for the number of packets and the number of non-full packets in Web flows. Most Web flows only include a small amount of packets in each direction. About 80% of the flows have less than 10 packets in the downstream direction. For the upstream direction, the number of packets inside a flow is even less. This observation is understandable as most Web objects are small and take only a few packets

to be transfered.

For the upstream direction, the two curves for the number of total packets and non-full packets almost overlap, indicating most upstream packets are not full. On the other hand, for the downstream direction, non-full packets are considerably less than the total packets. It is understandable as a Web object is normally carried by several full packets and one non-full packet at the end. The average packet size for upstream and downstream Web flows are also quite different as shown in Figure 9.7. Packet sizes for the upstream Web flows are generally small, while packet sizes for the downstream Web flows have a big range.

We show CCDFs for the percentages of non-full packets that can be piggybacked in Figure 9.8 and 9.9. Even under the host-to-host scope, over 70% Web flows can have part of their packets piggybacked and over 60% Web flows can have all their non-full packets piggybacked. Under the cluster-to-cluster scope, the percentages are about 15% higher. The curves for the upstream and downstream directions are similar. However, because there are less non-full packets in the downstream direction than the upstream direction, the number of packets that can be piggybacked is actually smaller for the downstream direction.

Most concurrent traffic of the Web flows are other Web flows. Under the cluster-to-cluster scope, a significant amount of DNS flows are also observed preceding the Web flows. Most of these flow relationships are caused by inherent application behavior. A Web session normally starts with a DNS query, then has several TCP connections established. In each connection, one or more Web objects may be retrieved. These rich and

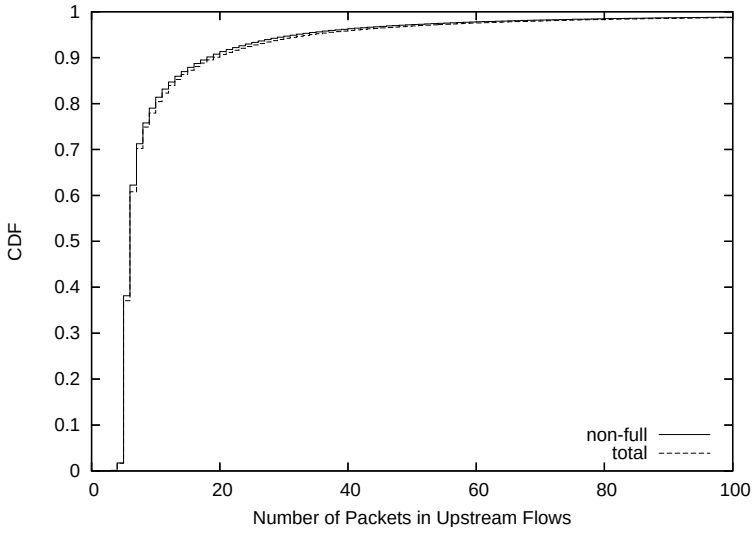


Figure 9.5: CDFs of the Number of Packets and the Number of Non-full Packets in Web Upstream Flows (for wpi1 log)

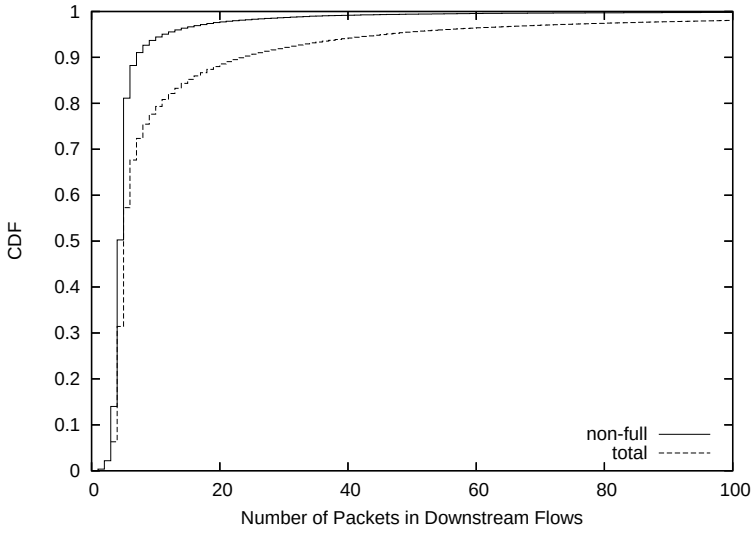


Figure 9.6: CDFs of the Number of Packets and the Number of Non-full Packets in Web Downstream Flows (for wpi1 log)

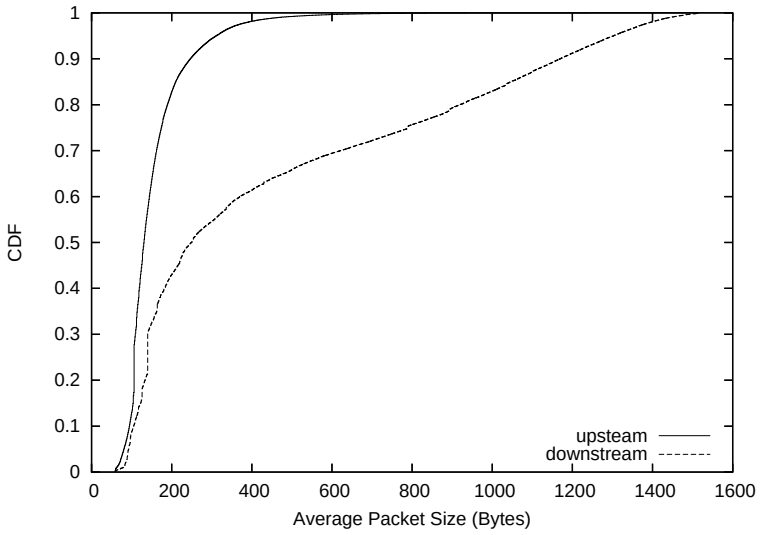


Figure 9.7: CDFs of Average Packet Sizes for Upstream and Downstream Web Flows (for wpi1 log)

inherent relationships grant two types of opportunities. First, important packets such as requests can be protected by letting related flows piggyback duplicate copies. Second, future flows or packets can be predicted given existing relationships between Web objects. These predicted packets can be piggybacked by other flows without introducing any extra transmissions. For example, the DNS-enabled Web (DEW) approach [KLR03] takes advantage of this type of relationship and lets DNS packets piggyback predicted Web messages.

9.2.3 Piggybacking for "Real" Streaming

A "real" streaming session normally includes two flows, one for control and one for data. For both flows, there are a list of ports to choose from. By default, the control flow uses the standard RTSP (Real Time Streaming

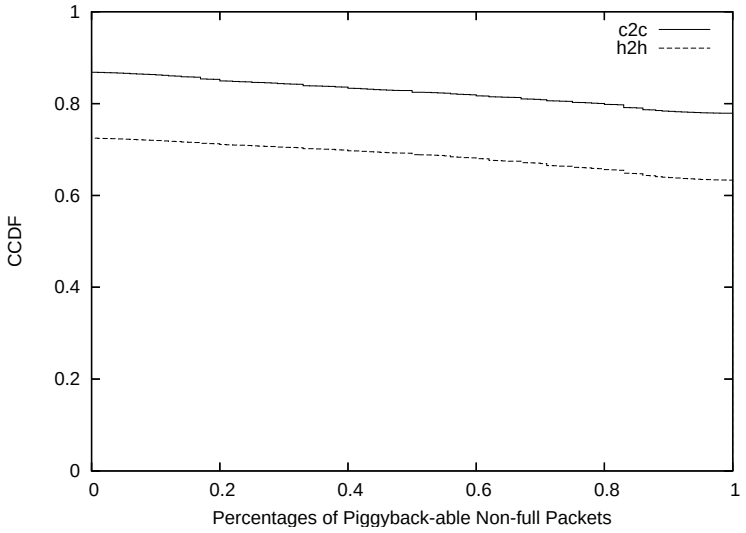


Figure 9.8: CCDFs of Percentages of Piggyback-able Non-full Packets in Web Upstream Flows (for wpi1 log)

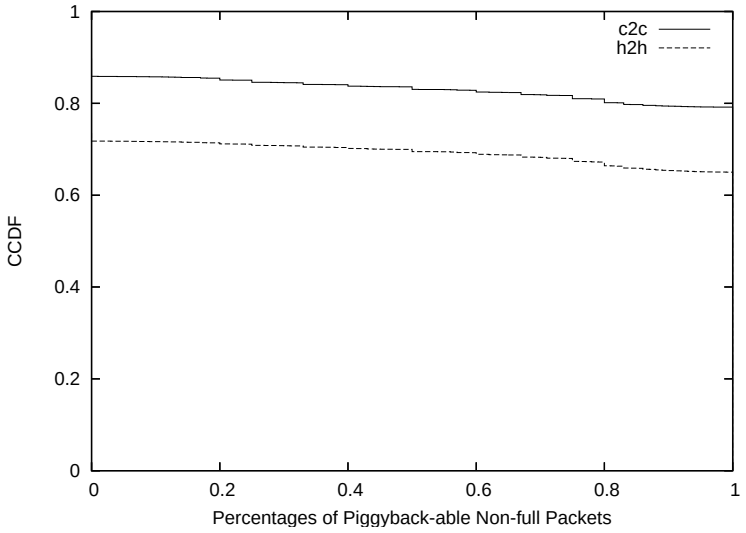


Figure 9.9: CCDFs of Percentages of Piggyback-able Non-full Packets in Web Downstream Flows (for wpi1 log)

Protocol) port (i.e. TCP port 554) and the data flow picks a UDP port between 6970 and 7170. If the connection is refused at the TCP port or no packet is received on the UDP port within a certain period, other ports are tried in turn. For simplicity, we use TCP port 554 to identify the "real" control flow and UDP port range 6970-7170 to identify the data flow. By this method, some "real" streaming flows may be missed if they are using different ports.

Using port numbers to identify "real" streaming flows may also cause a false positive problem, where a flow is identified as a "real" streaming flow but actually is not. "Real" streaming applications by default use UDP port range 6970-7170 for data transmission. But this port range is not reserved exclusively for that purpose. In order to distinguish "real" flows from others, we study the packet characteristics of "real" streaming flows. We manually collected packet traces for several "real" streaming sessions. We found that the average packet sizes for "real-audio" data flows are around 600 bytes and over 1000 bytes for "real-video" flows. In addition, most data flows are preceded by a RTSP flow (the control flow). Using the packet size and the relationship with RTSP flows as criteria, we found that most flows that have been identified as "real" data flows by port numbers in the isp1, isp2, and wpi3 logs are actually not. However for the wpi1 and wpi2 logs, over 90% of the identified data flows pass the test of the criteria.

We also consider the "real" data flows that have less than 10 packets in the downstream direction incomplete and discard them. The cutoff number is chosen based on our observation that even a few seconds of streaming incurs over tens of packets. There are just above 100 valid "real" flows

included in the isp1, isp2, and wpi3 logs. The wpi1 and wpi2 logs both have over a thousand valid "real" flows and they have similar results. We only show the set of results for the wpi1 log.

Figure 9.10 and 9.11 show the CDFs for the number of packets and the number of non-full packets in "real" data flows. We do not show the results for "real" control flows as we focus on examining the piggybacking opportunities for data packets. In the downstream direction, about 50% of flows have over 1K packets and a small amount of flows have over 10K packets. The curve for the non-full packets is close to that of total packets, indicating most packets are not full. The observation is the mixed results for audio and video flows. "Real" audio flows normally have packet size around 600 bytes, while "real" video flows have many full packets and some non-full packets. Not surprisingly, the upstream flows have fewer packets as these packets are only used to provide feedbacks.

The average packet size for each flow is shown in Figure 9.12. The packet sizes for upstream flows are small as these packets carry only feedback information. The majority of downstream flows have average packet sizes just above 600 bytes, indicating that most flows are audio flows.

We show CCDFs for the percentages of non-full packets that can be piggybacked in Figure 9.14 and 9.13. There are significant differences between the host-to-host and cluster-to-cluster scope. Under the host-to-host scope, most flows have opportunities to have a small amount of packets be piggybacked. The majority of the available space comes from their concurrent control flows. On the other hand, under the cluster-to-cluster scope, much space comes from another streaming data flow or a Web flow. As a result,

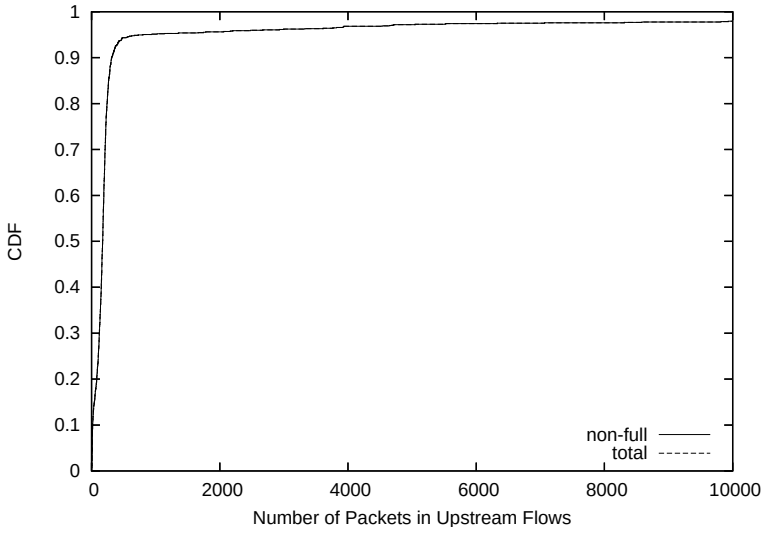


Figure 9.10: CDFs of the Number of Packets and the Number of Non-full Packets in "Real" Upstream Data Flows (for wpi1 log)

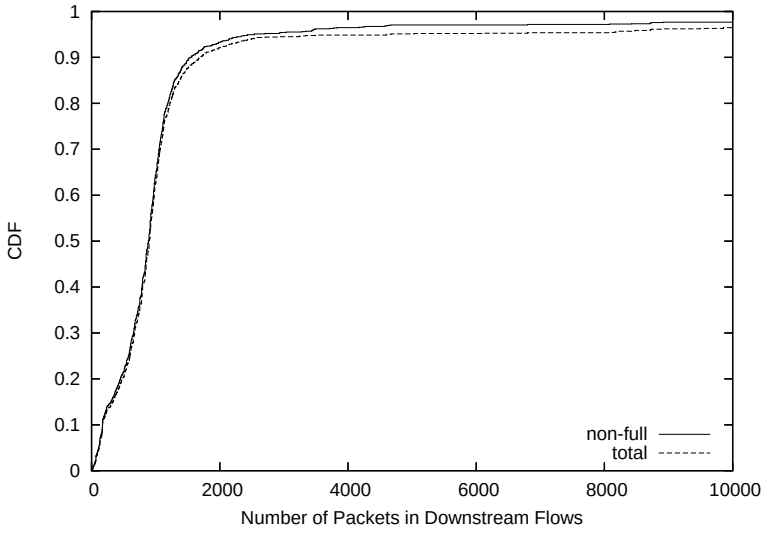


Figure 9.11: CDFs of the Number of Packets and the Number of Non-full Packets in "Real" Downstream Data Flows (for wpi1 log)

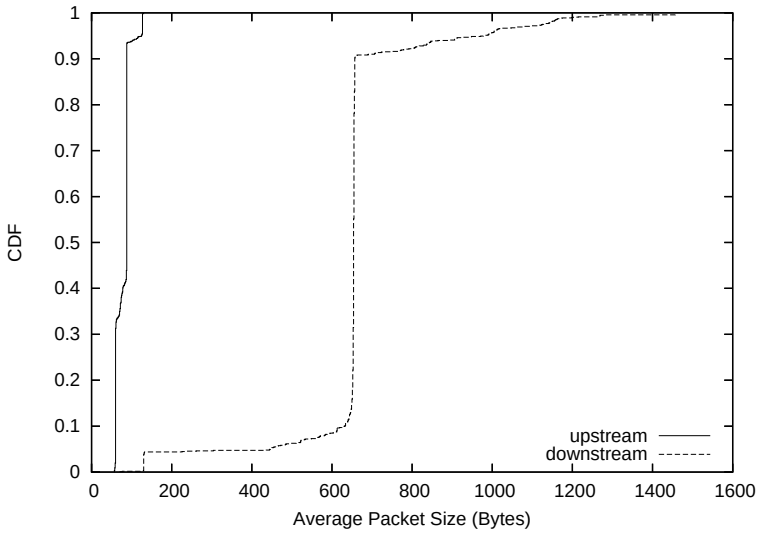


Figure 9.12: CDFs of Average Packet Sizes for Upstream and Downstream "Real" Data Flows (for wpi1 log)

significantly more packets can be piggybacked under the cluster-to-cluster scope.

Under the host-to-host scope, we do not see many piggybacking opportunities as available packet space from other flows is limited. However, under the cluster-to-cluster scope, a considerable amount of non-full packets can be piggybacked by other flows. While we did not study directly how critical packets inside streaming flows could be protected by sending redundant data, we show there are certain opportunities given much space provided by other flows within the same cluster pair.

9.2.4 Piggybacking for TCP Establishment

We have examined packet piggybacking possibilities for several particular applications. We now look at piggybacking possibilities for all TCP appli-

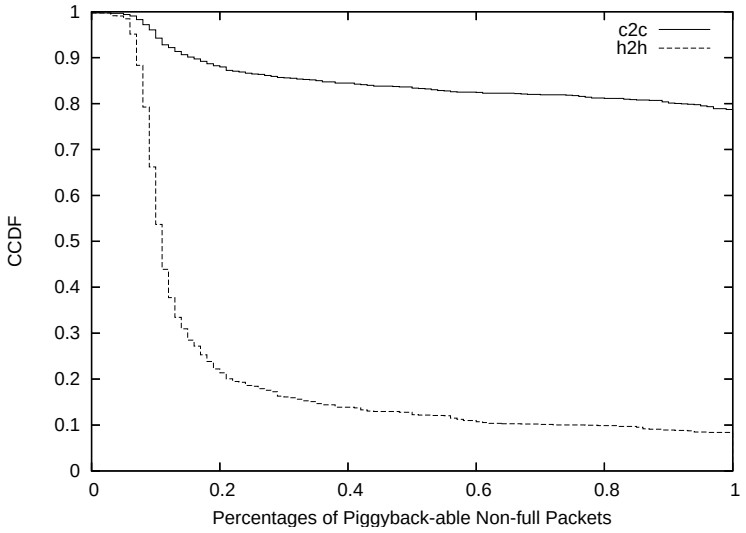


Figure 9.13: CCDFs of Percentages of Piggyback-able Non-full Packets in Upstream "Real" Data Flows (for wpi1 log)

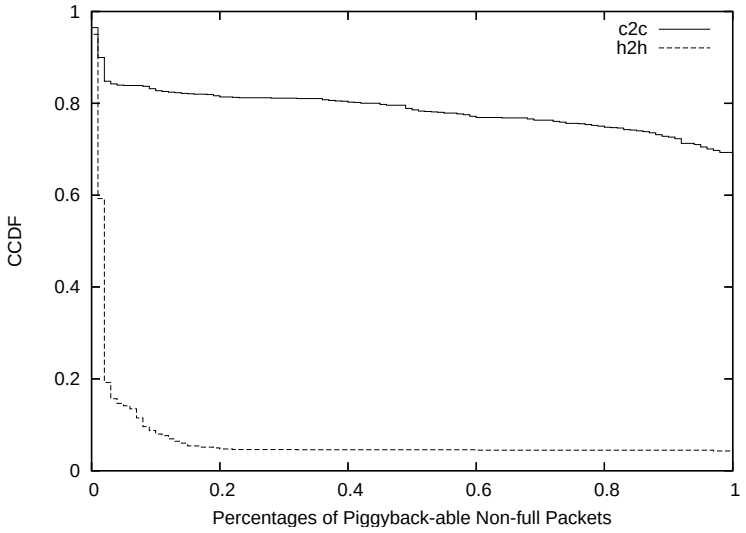


Figure 9.14: CCDFs of Percentages of Piggyback-able Non-full Packets in Downstream "Real" Data Flows (for wpi1 log)

cations. The TCP connection establishment procedure requires a three-step handshake. The loss of a SYN or SYN ACK packet causes a long timeout before a retransmission can take place. For both Linux and Windows systems, the default RTO for SYN packets is 3 seconds. This long timeout may cause serious performance problems in the presence of packet loss.

We can treat these handshake packets as another type of critical packets. They can be protected by letting a concurrent flow piggyback a duplicate copy after a small delay. As SYN and SYN ACK are both small, these duplicates require little space. We investigate the possible extent to which both the SYN and SYN ACK packets of a TCP flow can be piggybacked by other concurrent flows. We examine all TCP flows as well as TCP flows for some popular applications. The observations are consistent across all WPI logs while the results for ISP logs are lower under the cluster-to-cluster scope. We show the results for the wpi1 log in Figure 9.15.

For all TCP flows, about 40% of them can have their SYN and SYN ACK packets piggybacked by other flows in the host-to-host scope. This percentage goes up to over 70% under the cluster-to-cluster scope. We also list the possibilities for some popular applications. Most Web flows could have their TCP connection establishment protected. 98% of FTP data flows may have SYN packets piggybacked by their corresponding control flows or other concurrent data flows. Applications like msn-messenger, SSH, Telnet, SMTP, and RTSP show significant difference between the host-to-host scope and cluster-to-cluster scope. While there are not many opportunities for SYN packets to be piggybacked by other flows from the same host pairs, these applications have a much better chance by taking advantages

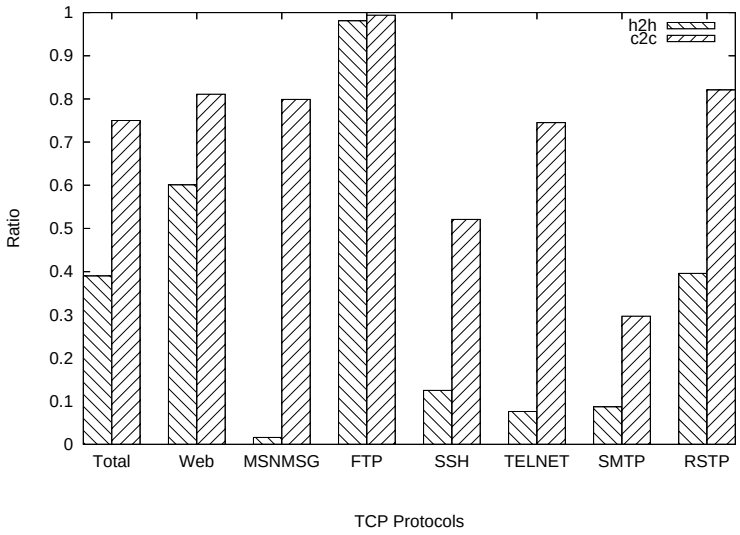


Figure 9.15: Possibility of SYN and SYN ACK Packets of TCP Flows Being Piggybacked by Other Flows (for wpil log)

of available space from flows within the same cluster pairs.

Note that duplicated SYN does not cause the SYN attack problem. SYN attack happens when an attacker sends a series of SYN requests with spoofed source IP addresses to a target attempting to use up the victim's resource. The SYN packets in a SYN attack are unique and the victim treats each of them as a request for a new connection. However, duplicated SYN packets are exactly the same. The server treats them for the same connection request and silently discards the duplicate if it already gets the original.

9.3 Summary

We have examined the potentials of critical packet piggybacking in four particular scenarios. Depending on the relationships with other flows and

the packet characteristics, an application could have different opportunities for its packets to be piggybacked. Web flows show the biggest potential as there are many non-full packets and plenty of flow relationships between Web and other flows. Performance-critical packets or even a whole flow can be protected by sending them in duplicate by other flows. In addition, due to the inherent relationships between flow contents, future packets or flows may be predicted. An application can indicate predicted traffic to be sent only if piggybacking is possible. When those predicted packets are successfully piggybacked, applications may experience better performance without extra transmissions. If they fail to find an opportunity of being piggybacked, the normal functionality of the application will not be influenced.

Critical packet piggybacking for SSH is not as promising as for Web flows. There are not many concurrent flows under the host-to-host scope. However, flow relationships become much richer under the cluster-to-cluster scope, which provides considerable opportunities to protect a certain amount of packets. As most packets inside a SSH flow are small, they can be easily piggybacked even only a small amount of space is available from other flows.

Reliability of the streaming application can be improved by sending redundant data for important packets. Under the host-to-host scope, the opportunities to piggyback these redundant data are small as there is little space provided by other concurrent flows. However, significantly more space is available if we also consider flows within the same cluster pairs. We have observed that in our logs over 70% of streaming flows could have

all of its packets protected. As the majority of the streaming flows in these logs are audio flows, we expect that this number will be smaller if video flows take more share.

The loss of SYN or SYN ACK packet can have a more negative impact than the loss of a regular packet in the middle of a TCP connection. In order to improve reliability, one approach is to send a duplicate SYN packet after the original SYN packet is sent. Our results show that over 40% of TCP flows could have these duplicate SYN packets be piggybacked by other concurrent flows. Under the cluster-to-cluster scope, the percentage goes up to 70%. Another similar usage of this approach is to send duplicate DNS requests with a shorter timeout than normal.

Chapter 10

Conclusion

10.1 Review of Motivation and Goals

Internet applications continue to have performance issues due to many reasons such as imperfect network conditions, inefficient transport protocols, and less optimal application designs. Increased bandwidth cannot help to reduce long round trip times, which are still a problem to applications that need fast request-response interactions. Even with current bandwidth over-provisioning in core networks and significantly improved access-link speed, packet loss is still unavoidable due to Internet traffic dynamics. Packet loss may cause serious performance problems to applications, especially when they incur long timeouts. Due to the nature of how current Internet applications are used, a large number of packets are small, which not only causes a problem for transmission efficiency, but also incurs much switching overhead to intermediate routers. All of these problems are challenges to current Internet application performance.

In this work, we seek to exploit flow relationships as a way to improve the performance of Internet applications. While there is much potential in this direction, little work has been done, and none in a systematic way. The goals of this thesis are to first have a better understanding of flow relationships; then to examine the potential improvements by using these relationships and establish a framework on possible techniques; finally to investigate specific techniques that exploit particular flow relationships and evaluate their offered improvements.

10.2 Results and Evaluation

10.2.1 Study on the Existence of Flow Relationship

In order to exploit flow relationships, the first question is to understand to what extent flow relationships exist and how much available space is provided by related flows. We answer this question in our background study in Chapter 3. We define a relationship to exist between two flows if packets inside the flows exhibit temporal proximity within the same scope, where the scope may either be between two hosts or between two clusters.

We examine the existence of flow relationships in five trace logs collected at two sites. Our results show the existence of a significant number of flow relationships. Under the host-to-host scope, we observed that 27-49% flows have concurrent flow relationships, in which one flow has temporal overlap with another flow. The percentages go up to 40-59% if we also include sequential flow relationships, in which one flow follows another within a 10-second threshold. We observed more flow relationships under

the cluster-to-cluster scope, where 44-77% flows show concurrent flow relationships and the percentage increases to 51-84% if also including sequential flow relationships within a 10-second threshold. By breaking down flows into different types based on their corresponding transport protocols and ports, we find that many relationships between different types of flows present relatively stable patterns.

We also study the characteristics of packet size and observed that many packets are small and that most packets do not reach the full Ethernet MTU size of 1500 bytes. This observation is consistent with other packet size studies conducted at different times and locations. We combine flow relationships and packet size characteristics to examine the existence of relationships between flows that have minimum percentages of non-full packets. Even with this constraint, the results still show the existence of many flow relationships as most flows have enough non-full packets.

The results on the flow relationship study are encouraging. Many flow relationships exist and most related flows have non-full packets. The unused transmission capacity provides potential for improving application performance. Given the existence of many flow relationships, the available packet space can be used by not only a flow itself, but also its related flows.

10.2.2 Framework of Exploiting Flow Relationships

Having examined the existence of flow relationships, we explore possible techniques that exploit flow relationships in Chapter 4. In this chapter, we established a framework on general approaches of exploiting flow relationships. One challenge of the generalization is that there are many Internet

applications. We need to abstract the common characteristics of the applications and apply techniques to types of applications instead of to particular applications. The other challenge is that there are also many ways to exploit flow relationships. We need to generalize the potential improvements that can be obtained by using flow relationships and explore particular techniques under each type of improvement.

Based on different traffic characteristics and performance concerns, we first categorize four types of stages that are commonly seen, including bulk transfer, interactive, transactional, and streaming. An application session may include only one stage or can be composed of multiple stages. By using this stage-based taxonomy, we look for general techniques that help a type of stage instead of a particular application.

We then examine the potential improvements that can be achieved by exploiting flow relationships. We categorize them as reducing total packets, providing better information, avoiding timeout, and reducing the number of RTTs. Under each category, we look at specific techniques that use two types of flow relationships. First, techniques such as data piggybacking, enhanced TCP ACKs, aggressive timeout, and packet prediction essentially use intra-flow relationships. Second, techniques such as packet aggregation, information sharing, critical packet piggybacking, and upcoming flow prediction exploit inter-flow relationships.

The framework is an integration of the stage-based taxonomy and the improvement categories. Under this framework, we seek techniques to achieve potential improvements to help particular types of application stages. For example, packet prediction is one way to reduce the number of RTTs.

It uses content relationships between flow packets and helps to improve latency performance of transactional stages. As another example, the enhanced ACK technique uses the packet space in the reverse direction of a flow to provide better information for the forward direction, which is most useful to stages that are sensitive to network dynamics such as bulk transfer or streaming. We investigate several specific techniques and evaluate their offered potential improvements thereafter.

10.2.3 Piggybacking Related Domain Names to Improve DNS Performance

This study is motivated by the observation that a local domain name server (LDNS) frequently sends more than one query to the same authoritative domain name server (ADNS) for different names within a short period of time. With the help of application-level information, we find it possible to predict subsequent queries based on the first query. At the same time, we find that most DNS response packets are small and allow additional DNS records to be attached. We hence propose the piggybacking related names (PRN) approach, which predicts future queries and piggybacks related domain name resolutions to the response to the first query.

Trace-base simulations show that more than 50% of cache misses can be reduced if the prediction is perfect and response packet space is plentiful. Realistic policies, using frequency and relevancy data for an ADNS, reduce cache misses by 25-40% and all DNS traffic by 20-35%. These percentages improve if we focus the policies on resource records with smaller ATTLs.

Reduced local cache misses helps to improve user-perceived DNS lookup latency. By piggybacking multiple answers in one response packet, the total number of queries and responses is also reduced, which alleviates the workload on both LDNSs and ADNSs.

Compared with other approaches that also address improving the local cache hit rate, our approach is novel. We explictly use the relationships among queries and allow an ADNS to push resolutions for the predicted names to the corresponding LDNS. The PRN approach reduces both first-seen misses as well as previously-seen misses while other approaches reduce just the latter. The cost of PRN is also low as it reduces the number of query and response packets while requiring no changes to the existing DNS protocol. We also show improved performance by combining the PRN approach with the renewal-based approaches to create the hybrid approaches that perform significantly better than each of their component approaches.

We use the PRN approach as an example to show how relationships between flow packets are used to reduce the total number of RTTs. While this approach is particularly to DNS lookup, it is extensible to similar situations where future packets are predictable and there is enough packet space to piggyback them.

10.2.4 Data Piggybacking

TCP is a widely used protocol on the Internet. Due to the asymmetric traffic pattern presented in a large number of TCP connections, there is a significant amount of ACK-only TCP packets on the current Internet. Data Piggybacking is to piggyback application-level data into ACK packets to

be sent in the reverse channel. This approach creates a clear primary and secondary direction of data flow within a TCP connection. Data are piggybacked on the reverse channel only when an ACK packet is generated.

This mechanism allows applications to send reverse-channel data without generating additional network packets or incurring new connections. Such a mechanism could be used by peers in a peer-to-peer environment where the transfer of desired content from peer A to peer B could simultaneously support the exchange of useful content via piggybacked transfer from B to A. Incentives of a p2p application such as BitTorrent encourage clients to exchange data tit-for-tat in both directions [Coh03], which can be done more efficiently with our approach.

This approach does require changes to the current TCP mechanism. However, by using a user-level modification to regular TCP client and server programs, we are able to estimate the potential benefits of the data piggybacking. We find that the reverse channel throughput can match the effective reverse bandwidth limit without negative effects on the forward channel throughput. The number of reverse channel packets generated to achieve this throughput is significantly less than a simple bidirectional transfer over one connection or two independent connections. Even in the case of asymmetric links such as a home DSL connection, data piggyback sizes up to a few hundred bytes per packet can be supported in either an application-only or TCP-level implementation without reducing the forward-channel throughput or having an appreciable effect on the number of the reverse-channel packets.

We use this approach as an example to show how the available packet

space in flows is used to improve the transmission efficiency. Data piggybacking uses the relationship between the ACK packets in the downloading direction and the data packets in the uploading direction. When this relationship is properly used, we see the potential for reducing the total number of packets without impacting application performance.

10.2.5 TCP Enhanced ACKs

TCP enhanced ACKs is another way to exploit the available space provided in reverse-channel ACKs. Different from data piggybacking, the objective of TCP enhance ACKs is to provide better quality of information for applications. The current TCP acknowledge mechanism along with the traditional timestamp option only allow TCP to roughly measure the path conditions in the forward direction. The measurement is not only influenced by the congestion of the reverse direction, but also by the widely used TCP delayed ACK mechanism.

We propose a new TCP option that provides more detailed and complete information about the reception of data packets at the receiver compared with the existing TCP Timestamp Option [JBB92]. This information allows a TCP sender to track the spacing between all data packets arriving at the receiver and to have complete timing information for the forward and reverse directions of the connection. Such information can be used to better detect jitter and congestion in the forward direction than what is currently available.

Using the new TCP timestamp option, we can explicitly calculate the spacing between data packets received at the receiver and hence establish

jitter in the forward data transmission channel. Currently this spacing can only be inferred from the spacing between ACK generation at the receiver or ACK reception at the sender. Results from many file transfer experiments show discrepancies between the spacing of ACKs and the actual spacing of received data packets. These differences occur more frequently and with a larger magnitude when traffic is not sent as frequently as allowed by TCP as can be the case with Web or streaming data. In addition, the new option allows for a more accurate RTT estimation, which is not influenced by the factors of ACK delay or packet loss. Our results show that the current mechanism of RTT calculation may overestimate by more than 200ms when packet loss occurs. This discrepancy not only causes a more conservative determination of RTO, but also make it difficult for situations where it is desirable to decouple RTT from packet loss such as paths involving wireless subnets.

The TCP enhanced ACKs technique is an example of how available packet space is used to provide better quality of information for applications. An important point concerning the availability of this more complete and accurate information about the TCP connection is that it is obtained with minimal, bounded overhead by the TCP receiver and only a small amount of additional information is added to an ACK packet that must be sent anyways.

10.2.6 Packet Aggregation

Packet aggregation is encouraged by the observation of lots of non-full packets on the Internet and many concurrent flows existing between the

same host or cluster pairs. The assumption is that if we can aggregate small packets going to the same destination, we can save the total number of packets on the Internet. Reducing the total number of packets helps to alleviate the workload of intermediate routers whose processing cost is packet-based instead of byte-based. By sharing the common IP headers, a fraction of overhead can be reduced, therefore improving the bandwidth usage efficiency.

The results from a trace-based simulation are promising when we apply aggregation to all packets regardless to which flow they belong. There is about 20% packet savings if we use a small aggregation delay of 50ms and this number goes up to 50% if we use a larger aggregation delay of 1 second. However, problems exist in aggregation within the same flow. One example is the ACK compression problem and another is the packet dependency of a flow. Once we apply a constraint that requires aggregation occur only between packets from different flows, the packet savings are significantly reduced. There is only about 3%-6% packet savings for one log set in which a lot of P2P traffic is observed. For another log set in which P2P traffic is little, the packet savings are considerably bigger, but still marginal as only 10%-20% packet reduction is observed.

While packet aggregation does not gain much due to current Internet traffic patterns, we still consider it as a good example on how flow relationships can be used to improve the transmission efficiency. If an application allows for more flexibility in terms of when a packet may be sent (e.g. by giving a deadline), the gain of aggregation can be achieved without impacting the performance of the application.

10.2.7 Critical Packet Piggybacking

Packet loss may cause a severe problem to the application performance if the loss incurs a long timeout delay. We often observe this situation occurring in interactive or transactional stages of an application. For example, a DNS retransmission will occur in several seconds when an original request gets lost. The loss of TCP SYN or SYN ACK packets causes a 3 seconds delay before another try. We propose a scheme called *critical packet piggybacking*, which protects critical packets from loss by exploiting available packet space in other concurrent flows. The essence of this scheme is to let other flows piggyback a duplicate copy for packets that are critical to application performance. The transmission of the duplicate data only occurs if piggybacking is possible, therefore no extra packets are introduced.

We examine four particular scenarios in which critical packet piggybacking can be useful. We look at "SSH" and Web applications as they usually involve interactive or transactional stages which are sensitive to packet loss. We also examine the "real" streaming application as it represents a type of applications whose decoding algorithm is more sensitive to the loss of one set of frames than that of others. Finally, we check the possibility to protect packets for TCP establishment.

For these four scenarios, we investigate the possible extent to which the non-full packets in one flow can be piggybacked by other flows. We find that Web flows have the biggest potential in terms of piggybacking as there are many non-full packets and plenty of flow relationships for Web flows. A considerable percentage of SSH flows can also have a certain amount of

packets to be sent in duplicates by other flows under the same cluster pair. "Real" streaming flows do not show many opportunities for piggybacking under the host-to-host scope, but the opportunities increase significantly under the cluster-to-cluster scope. Finally, it is highly likely that the TCP SYN and SYN ACK packets can be protected by letting other concurrent flows send duplicate copies.

Critical packet piggybacking is another way to exploit packet space in related flows. Different from the objective of packet aggregation, critical packet piggybacking protects performance-critical packets from loss, therefore avoiding long timeouts. We find that this scheme has much potential for applications that involve interactive or transactional stages. First, the performance of these stages is vulnerable to packet loss. Second, packets in these stages are normally small and can be easily piggybacked by other concurrent flows.

10.3 Examination of the Hypothesis and Summary of Contributions

The results and evaluation support the hypothesis made in Section 1.4, i.e., "classes of techniques can be deployed to exploit flow relationships and enhance application performance with minimal costs." First, the existence of numerous flow relationships suggests much potential in exploring flow relationships. Second, the establishment of the framework indicates the general usefulness of exploiting flow relationships. Third, the investigation of specific techniques illustrates that a variety of improvements to Internet ap-

plications can be achieved with minimal costs by exploiting particular flow relationships. All of the above evidence supports the hypothesis that exploiting flow relationships is an effective direction to improve application performance.

We summarize the contributions of this work as following:

A systematic study of flow relationships: to our best knowledge, it is the first time that relationships between network flows are studied explicitely and in a systematic way. We studied the overall flow relationships as well as individual relationships between particular flow types. The scope of our study is for not only host pairs, but also cluster pairs. By combining packet size characteristics and flow relationships, we also investigated the available packet space in related flows. The results can be directly exploited to improve application performance, or used as a guideline for a better development of applications and protocols. Our study is beneficial to application performance in both cases.

A general framework on exploiting flow relationships: with a better understanding of the flow relationships, we realized that there is also a lack of a general framework on how flow relationships can be exploited. We hence established such a framework by using two sets of categorization. We classify applications by using a stage-based taxonomy and categorize techniques by their potential improvements. The framework combines the two categorizations together, which explicitly gives expected benefits and the applicable stages for a partic-

ular technique. This framework is important as it generalizes possible techniques that exploit flow relationships. It facilitates a specific technique to be extended to fit for a group of applications. The framework also helps a particular application to find the most appropriate techniques that meet its performance concerns.

A mechanism of application-level prediction: we proposed and evaluated a mechanism that predicts and piggybacks related domain names. Our experiment results show that this mechanism can significantly reduce local DNS cache miss rates, while at the same time reducing the same amount of DNS query and response messages. While this mechanism improves performance of a particular application, it can be applied to similar situations in which packet relationships can be exploited to predict future traffic and the available packet space can be used to piggyback this predicted information.

A method to efficiently use the reverse-channel of TCP connections: we proposed and evaluated the "data piggybacking" method, which uses the packet space provided in ACK packets of the reverse-channel to send data. This method does not influence the performance of the forward direction of a TCP connection, while it can provide transmission capacity on the reverse direction without introducing additional packets in the ideal case. Compared to the traditional TCP transmission mechanism, this method can reach compatible throughput for paths with symmetric bandwidth, but is more efficient in the number of generated packets. In the case of asymmetric links, data piggy-

backing can provide even better throughput than the traditional TCP transmission mechanism when data are sent blindly in both directions.

A new TCP option to provide better information: we proposed and evaluated a new TCP option which extends the current TCP Timestamp option. This new option allows explicit calculation of one-way delay jitter, which can only be estimated by the current TCP mechanism. Experiments show that the discrepancy between the inferred and the real measurement can be significant under certain situations. The new timestamp option also makes it possible to have a more accurate estimation of RTTs, which is decoupled from the packet loss and the delayed ACK mechanism. All these benefits can be obtained by adding only a few bytes to the current ACK packets.

An evaluation of packet aggregation: packet aggregation was proposed in a previous study, but its benefits were only evaluated under the assumptions that are not realistic. We used real traffic traces as well as a more realistic simulation model to examine the gains of packet aggregation. Under the constraint of aggregation for inter-flow packets only, the gain of packet savings is marginal. While we do not see this direction as appropriate to further investigate under current Internet traffic patterns, we provided valuable results of the possible benefits that can be gained by the scheme.

An approach to avoid timeouts: we proposed to use critical packet piggybacking to protect performance-critical packets from loss. We exam-

ined four particular scenarios and investigated the possibilities for packets in the corresponding applications to be piggybacked by other concurrent flows. The results show significant potential for these scenarios, especially for applications that have plenty of flow relationships and generate many small packets. While we did not explicitly measure the time saving of this scheme by avoiding timeouts, we did show that critical packet piggybacking has the potential to protect packets from loss for a minimum cost.

10.4 Future Directions

In this thesis, we presented that exploiting flow relationships is an valuable direction for improving application performance. Besides a number of techniques that we have explored, there remains much future work in this direction as outlined below:

Continuation of examining flow relationships: while we have studied flow relationships in a systematic way, we have not explicitly examined flow relationships for all applications. Applications such as P2P are not extensively studied in this thesis. We also expect that future applications may introduce new patterns of flow relationships. Continuation of examining flow relationships is a straightforward direction for future work.

Evaluation on broader data sets: we have evaluated our approaches in two sets of traffic traces in this study. With the establishment of the use-

fulness of these approaches, it will be interesting to see how they perform for a broader range of data sets. Examination and understanding how different traffic patterns influence these approaches will help to improve them.

Investigation of other techniques in the framework: while we have established a framework of potential improvements by exploiting flow relationships, the techniques discussed in this thesis are still only a part of the overall framework. Exploring and investigating other techniques to fill the framework is clearly a direction for future work.

Improvement of the PRN approach: an obvious direction for future work on the PRN approach is to examine alternate policies such as ones to consider the ATTL for an entry. Policies should also be tested with additional logs. Another direction of future work is to deploy the PRN approach at an ADNS. We expect it should perform better than our simulation because an ADNS has more complete knowledge of its site contents and it can also aggregate reference patterns from a greater number of clients. An ADNS will not know if predicted names are actually used, but it can detect and modify its piggybacked list as it learns new access patterns that could have been predicted. A final direction to explore with this approach is the different types of sites and contents for which it is most useful. Sites with few servers and long authoritative TTLs likely do not need improvement in DNS performance while we expect more dynamic sites would be the first to benefit from this approach.

Improvement of the data piggybacking approach: first, the data piggyback mechanism needs to be implemented and tested. Second, for asymmetric connections, too much data transmission in the reverse direction negatively affects forward-direction throughput. Mechanisms for the data piggybacking approach to monitor and adjust to available bandwidth limits need to be investigated. Another direction that was not explored in this work is how these approaches would work for network connections incorporating different types of wireless connectivities. On one hand increasing the size of packets beyond the minimum amount needed is a bad idea because it could increase the packet loss rates due to transmission errors. On the other hand, our results show that data piggybacking can yield comparable reverse-channel throughput comparing to more conventional approaches while generating fewer reverse channel packets, which would be a potential improvement in the face of higher loss rates. Investigation of these types of tradeoffs is a clear direction for future work.

Improvement of enhanced ACKs: future work includes the implementation of the enhanced Timestamp Option in the current TCP/IP stack. Another direction is to investigate potential improvements to TCP implementations using the more detailed and complete timestamp information. Another direction to explore is to understand how this enhanced information can help applications to make decisions. Currently in Linux systems, an application can get network information from the transport layer. This information can be extended to include

the information provided by the new TCP Timestamp Option such as the one-way delay jitter and more accurate RTT. Exploring how this enhanced information can help applications such as streaming is certainly an interesting direction for future work.

Exploration of additional use of TCP ACKs: another interesting direction is to explore additional uses for the unused capacity in the generated TCP ACK packets beyond what we have investigated in this work. Given the volume of such packets in the Internet, any improvements have much potential for making an impact. For example, while currently not allowed in TCP, it is possible to consider TCP options that are not constrained by the 40-byte limit in the existing TCP standard. It could be possible to propose a "fat" option that is reserved for use only when an ACK-only packet is generated by a receiver where the traditional data portion of the packet could be used for control information intended for the TCP-layer at the sender. This TCP-layer would know that the packet contains control information based on the kind of TCP option and know to use the control information rather than pass it up as data to an application.

Improvement of critical packet piggybacking: a direction of future work is to investigate how critical packet piggybacking behaves under different packet loss conditions. We expect that this scheme is most useful for error-prone paths. Another direction is to investigate the policies for applications or transport layers to decide which packets are critical. A third direction is to implement a piggybacking mech-

anism that allows applications to explicitly specify a deadline for a packet to be piggybacked. If the sending request is not fulfilled before the deadline, the application needs to be notified of the failure. The mechanism should also allow an application to revoke a sending request if the packet is still in the queue.

10.5 Summary

This thesis is a coherent piece of work, which covers a systematic study of flow relationships, a general framework of exploiting flow relationships, and specific techniques that exploit particular flow relationships and show different types of improvements. Our experimental results show positive support for the hypothesis we made in Section 1.4, i.e., "classes of techniques can be deployed to exploit flow relationships and enhance application performance with minimal costs."

In this work, we have shown that exploiting flow relationships is a useful direction for improving application performance. Given the existence of numerous flow relationships and various exploitation techniques, we see that exploration in this direction is fruitful and it leads to much future work.

Bibliography

[AER98] Hossam Afifi, Omar Elloumi, and Gerardo Rubino. A dynamic delayed acknowledgment mechanism to improve TCP performance for asymmetric links. In *Proceedings of the IEEE Symposium on Computers and Communications*. IEEE, June/July 1998.

[AP99] Mark Allman and Vern Paxson. On estimating end-to-end network path properties. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 263–274, New York, NY, USA, 1999.

[APS99] M. Allman, V. Paxson, and W. Stevens. TCP congestion control, April 1999. RFC 2581.

[arg] Argus - IP network auditing facility. `http://www.qosient.com/argus`.

[BBC+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services, December 1998. RFC 2475.

[BC98] Paul Barford and Mark Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 151–160, Madison, WI, 1998.

[BCS94] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview, June 1994. RFC 1633.

[BDH99] D. Borman, S. Deering, and R. Hinden. IPv6 Jumbograms, August 1999. IETF RFC 2675.

[BDJ01] Supratik Bhattacharyya, Christophe Diot, and Jorjeta Jetcheva. Pop-level and access-link-level traffic dynamics in a tier-1 pop. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, 2001.

[BG99] Robert Buff and Arthur Goldberg. Web servers should turn off Nagle to avoid unnecessary 200 ms delays, April 1999. `http://www.cs.nyu.edu/artg/research/speedingTCP/buff_goldberg_speeding_up_TCP.ps`.

[BP95] Lawrence S. Brakmo and Larry L. Peterson. TCP Vegas: End to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, October 1995.

[BPFS02] H. Balakrishnan, V.N. Padmanabhan, G. Fairhurst, and M. Sooriyabandara. TCP performance implications of network path asymmetry, December 2002. RFC 3449.

[BPK99] Hari Balakrishnan, Venkata Padmanabhan, and Randy H. Katz. The effects of asymmetry on TCP performance. *Mobile Networks and Applications*, 4:219–241, 1999.

[BPS+98] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, M. Stemm, and R.H. Katz. TCP Behavior of a Busy Internet Sever: Analysis and Improvements. In *Proceedings of the IEEE Infocom 1998 Conference*. IEEE, March 1998.

[Bra94] R. Braden. T/TCP – TCP Extensions for Transactions Functional Specification, July 1994. RFC 1644.

[BRS99] H. Balakrishnan, H. S. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proceedings of the ACM SIGCOMM 1999 Conference*, pages 175–187. ACM, September 1999.

[BS99] B. R. Badrinath and Pradeep Sudame. Car pooling on the Net: Performance and implications. Technical report, Rutgers University, May 1999.

[BS00] B.R. Badrinath and Pradeep Sudame. Gathercast: The design and implementation of a programmable aggregation mechanism for the internet. In *Proceedings of the IEEE International*

Conference on Computer Communications and Networks, October 2000.

[CDJM91] R. Caceres, P. B. Danzig, S. Jamin, and D. J. Mitzel. Characteristics of wide-area tcp/ip conversations. In *Proceedings of ACM SIGCOMM '91*, pages 101–112, Zurich, Switzerland, September 1991.

[CER] CERT/CC. Vulnerability Note VU109475. `http://www.kb.cert.org/vuls/id/109475`.

[CK01] Edith Cohen and Haim Kaplan. Proactive caching of DNS records: Addressing a performance bottleneck. In *Proceedings of the Symposium on Applications and the Internet*, pages 85–94, San Diego-Mission Valley, CA, USA, January 2001. IEEE-TCI.

[CK02] Edith Cohen and Haim Kaplan. Prefetching the means for document transfer: A new approach for reducing web latency. *Computer Networks*, 39(4):437–455, July 2002.

[cLH03a] Kun chan Lan and John Heidemann. On the correlation of internet flow characteristics. Technical Report Technical Report ISI-TR-574, USC/Information Sciences Institute, July 2003. `http://www.isi.edu/~johnh/PAPERS/Lan03c.html`.

[cLH03b] Kun chan Lan and John Heidemann. A tool for rapid model parameterization and its applications. In *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, Karlsruhe, Germany, August 2003.

[Coh03] Bram Cohen. Incentives build robustness in BitTorrent, May 2003. `http://www.bittorrent.com/bittorrentecon.pdf`.

[Con] Internet Software Consortium. BIND DNS Server. `http://www.isc.org/products/BIND/`.

[CSA00] N. Cardwell, S. Savage, and T. Anderson. Modeling tcp latency. In *Proceedings of the IEEE Infocom 2000 Conference*, Tel-Aviv, Israel, March 2000. IEEE.

[CT90] D.D. Clark and D.L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. *ACM Computer Communication Review*, 20(4):200–208, September 1990.

[CV01] Girish P. Chandranmenon and George Varghese. Reducing web latency using reference point caching. In *Proceedings of IEEE Infocom 2001*, pages 1607–1616, 2001.

[DVM+03] Gali Diamant, Leonid Veytser, Ibrahim Matta, Azer Bestavros, Mina Guirguis, Liang Guo, Yuting Zhang, and Sean Chen. itmBench: Generalized API for Internet Traffic Managers. Technical Report BU-CS-2003-032, CS Department, Boston University, Boston, MA 02215, December 2003.

[EHT00] L. Eggert, J. Heidemann, and J. Touch. Effects of Ensemble-TCP. *ACM Computer Communication Review*, 30(1):15–29, January 2000.

[EV01] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 75–80, 2001.

[FB02] Nick Feamster and Hari Balakrishnan. Packet Loss Recovery for Streaming Video. In *12th International Packet Video Workshop*, Pittsburgh, PA, April 2002.

[FCL01] Chengpeng Fu, Ling Chi Chung, and Soung C. Liew. Performance degradation of TCP Vegas in asymmetric networks and its remedies. In *Proceedings of the IEEE International Conference on Communications*, June 2001.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, June 1999. RFC 2616.

[FH99] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm, April 1999. RFC 2582.

[FJ93] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[FKMkc03] M. Fomenkov, K. Keys, D. Moore, and k claffy. Longitudinal study of Internet traffic from 1998-2003: a view from 20 high performance sites. Technical report, Cooperative Association for Internet Data Analysis (CAIDA), April 2003.

[FML+03] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet-level traffic measurements from the Sprint IP backbone. *IEEE Network*, 2003.

[FP99] W. Fang and L. Peterson. Inter-as traffic patterns and their implications. In *IEEE Global Internet Symposium*, December 1999.

[GM02a] Luigi Alfredo Grieco and Saverio Mascolo. Tcp westwood and easy red to improve fairness in high-speed networks. In *PIHSN '02: Proceedings of the 7th IFIP/IEEE International Workshop on Protocols for High Speed Networks*, pages 130–146, 2002.

[GM02b] Liang Guo and Ibrahim Matta. Differentiated Control of Web Traffic: A Numerical Analysis. In *Proceedings of SPIE ITCOM'2002: Scalability and Traffic Control in IP Networks*, Boston, MA, August 2002.

[GM04] Luigi A. Grieco and Saverio Mascolo. Performance evaluation and comparison of westwood+, new reno, and vegas tcp congestion control. *SIGCOMM Comput. Commun. Rev.*, 34(2):25–38, 2004.

[GN98] Jim Gettys and H. F. Nielsen. SMUX Protocol Specification, July 1998. Work In Progress (W3C Working Draft WD-mux-19980710).

[GSG02] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *Proceedings of the Second ACM SIGCOMM Internet Measurment Workshop*, Marseille, France, 2002.

[Han96] Charles M. Hannum. Security Problems Associated With T/TCP, September 1996. `http://tcp-impl.grc.nasa.gov/tcp-impl/list/archive/1292.html`.

[JBB92] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance, May 1992. RFC 1323.

[JC03] Baekcheol Jang and Kilnam Chon. DNS resolution with renewal using piggyback. In *Proceedings of the Twelfth International World Wide Web Conference (Poster)*, Budapest, Hungary, May 2003.

[JSBM02] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. Dns performance and the effectiveness of caching. *IEEE/ACM Transactions on Networking*, 10(5):589–603, October 2002.

[KLR03] Balachander Krishnamurthy, Richard Liston, and Michael Rabinovich. DEW: DNS-enhanced web for faster content delivery. In *Proceedings of the Twelfth International World Wide Web Conference*, Budapest, Hungary, May 2003.

[KS04] Srisankar S. Kunniyur and R. Srikant. An adaptive virtual queue (avq) algorithm for active queue management. *IEEE/ACM Trans. Netw.*, 12(2):286–299, 2004.

[KSO+01] Jed Kaplan, P.J. Singh, Mike O'Dell, John Hayes, Ted Schroeder, Daemon Morrell, and Jennifer Hsu. Extended Ethernet Frame Size Support, November 2001. http://www.ietf.org/proceedings/03nov/I-D/draft-ietf-isis-ext-eth-01.txt.

[KVR98] Lampros Kalampoukas, Anujan Varma, and K.K. Ramakrishnan. Two-way TCP traffic over rate controlled channels: Effects and analysis. *IEEE/ACM Transactions on Networking*, 6(6):729–743, December 1998.

[KW99] H.T. Kung and S.Y. Wang. TCP trunking; design, implementation, and performance. In *Proceedings of IEEE ICNP 1999 Conference*. IEEE, November 1999.

[KW00] Balachander Krishnamurthy and Jia Wang. On network-aware clustering of web clients. In *Proceedings of the ACM SIGCOMM '00 Conference*, Stockholm, Sweden, August 2000.

[LC00] Y. Liu and M. Claypool. Using Redundancy to Repair Video Damaged by Network Data Loss. In *ACM Multimedia Computing and Networking (MMCN)*, San Jose, CA, Jan 2000.

[LCK02] Mingzhe Li, Mark Claypool, and Robert Kinicki. MediaPlayer versus RealPlayer – A Comparison of Network Turbulence. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop (IMW)*, pages 131 – 136, Marseille, France, November 2002.

[LCKN04] Mingzhe Li, Mark Claypool, Robert Kinicki, and James Nichols. Characteristics of Streaming Media Stored on the Web. *ACM Transactions on Internet Technology (TOIT)*, 2004. (Accepted for publication).

[LSZ02] Richard Liston, Sridhar Srinivasan, and Ellen Zegura. Diversity in DNS performance measures. In *Proceedings of the Second ACM SIGCOMM Internet Measurement Workshop*, pages 19–31, Marseille, France, 2002.

[MB00] Ibrahim Matta and Azer Bestavros. QoS Controllers for the Internet. In *Proceedings of the NSF Workshop on Information Technology*, Cairo, Egypt, March 2000.

[MC00] S. McCreary and K. Claffy. Trends in wide area IP traffic patterns: A view from Ames Internet Exchange. In *Proceedings of the ITC Specialist Seminar on IP Traffic Modeling, Measurement and Management*, September 2000.
`http://www.caida.org/outreach/ papers/AIX0005/AIX0005.pdf.`

[MCG+01] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 287–297, 2001.

[MD90] J. Mogul and S. Deering. Path MTU Discovery, November 1990. IETF RFC 1191.

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options, October 1996. RFC 2018.

[Moc87a] P. Mockapetris. Domain Names - Concepts and Facilities, November 1987. RFC 1034.

[Moc87b] P. Mockapetris. Domain Names - Implementation and Specification, November 1987. RFC 1035.

[Mog92] Jeffrey C. Mogul. Observing tcp dynamics in real networks. In *SIGCOMM '92: Conference proceedings on Communications architectures & protocols*, pages 305–317, 1992.

[MPFL96] Joan L. Mitchell, William B. Pennebaker, Chad E. Fogg, and Didier J. Legall, editors. *MPEG Video Compression Standard*. Chapman & Hall, Ltd., London, UK, UK, 1996.

[Nag84] J. Nagle. Congestion Control in IP/TCP internetworks, January 1984. RFC 896.

[NLA] NLANR. network traffic packet header traces. `http://pma.nlanr.net/Traces/`.

[OMP02] D. Ott and K. Mayer-Patel. Transport-level Protocol Coordination in Cluster-to-Cluster Applications. In *Proceedings of 2002 USENIX Annual Technical Conference*, pages 147–159, June 2002.

[Pad99] V. Padmanabhan. Coordinated congestion management and bandwidth sharing for heterogeneous data streams. In *Proceedings of NOSSDAV 1999 Conference*, pages 187–190, June 1999.

[PCN00] P. Pradhan, T. Chiueh, and A. Neogi. Aggregate TCP congestion control using multiple network probing. In *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS2000)*, April 2000.

[Per96] C. Perkins. IP Encapsulation within IP, October 1996. IETF RFC 2003.

[por] Port Numbers Specified by Internet Assigned Numbers Authority. `http://www.iana.org/assignments/port-numbers`.

[PPPW04] KyoungSoo Park, Vivek S. Pai, Larry Peterson, and Zhe Wang. CoDNS: Improving DNS performance and reliability via cooperative lookups. In *Symposium on Operating Systems Design and Implementation*, San Francisco, CA, December 2004.

[PTB+02] K. Papagiannaki, N. Taft, S. Bhattacharyya, P. Thiran, K. Salamatian, and C. Diot. A pragmatic definition of elephants in internet backbone traffic. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, 2002.

[QZK99] L. Qiu, Y. Zhang, and S. Keshav. On individual and aggregate tcp performance. In *Proceedings of ICNP*, Toronto, Canada, November 1999.

[RASHB02] Lopa Roychoudhuri, Ehab Al-Shaer, Hazem Hamed, and Greg Brewster. On studying the impact of the internet delays on audio transmission. In *Proceedings of the IEEE Workshop on IP Operations and Management*, October 2002.

[RSSD04] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 135–148. ACM Press, 2004.

[SAA+99] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: Informed internet routing and transport. *IEEE Micro*, 19(1):50–59, January 1999.

[SB04] Joel Sommers and Paul Barford. Self-configuring network traffic generation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 68–81, Sicily, Italy, October 2004.

[SK02a] Pasi Sarolahti and Alexey Kuznetsov. Congestion Control in Linux TCP. In *Proceedings of 2002 USENIX Annual Technical Conference, Freenix Track*, pages 49–62, Monterey, CA, June 2002.

[SK02b] Pasi Sarolahti and Alexey Kuznetsov. Congestion control in linux tcp. In *Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference*, Berkeley, CA, USA, 2002. USENIX Association.

[Spe] S. Spero. Session Control Protocol, Version 1.1. `http://www.w3.org/Protocols/HTTP-NG/http-ng-scp.html`.

[SPH05] Rishi Sinha, Christos Papadopoulos, and John Heidemann. Internet packet size distributions: Some observations, October 2005. `http://netweb.usc.edu/~rsinha/pkt-sizes/`.

[Spr04] Sprint IP monitoring project, packet trace analysis, February 2004. `http://ipmon.sprint.com/packstat/packetoverview.php`.

[SRB01] Shriram Sarvotham, Rudolf Riedi, and Richard Baraniuk. Connection-level analysis and modeling of network traffic. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, 2001.

[SRS99] Anees Shaikh, Jennifer Rexford, and Kang G. Shin. Load-sensitive routing of long-lived ip flows. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 215–226, 1999.

[Ste97] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, January 1997. IETF RFC 2001.

[SXM+00] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, K. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol, October 2000. IETF RFC 2960.

[tcpa] Tcpdump/libpcap. `http://www.tcpdump.org/`.

[tcpb] Unix man pages : tcp (7).

[TMW97] K. Thompson, G. J. Miller, and R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics. *IEEE Network*, 11:10–23, November 1997.

[Tou97] J. Touch. TCP Control Block Interdependence, April 1997. RFC 2140.

[Tur05] Bryan Turner. Generalizing bittorrent: How to build data exchange markets (and profit from them!), January 2005. `http://www.fractalscape.org/GeneralizingBitTorrent.htm`.

[Vix99] P. Vixie. Extension Mechanisms for DNS (EDNS0), August 1999. RFC 2671.

[VLL05] Bryan Veal, Kang Li, and David Lowenthal. New methods for passive estimation of TCP round-trip times. In *Proceedings of the Passive and Active Measurement Workshop*, Boston, Massachusetts, March/April 2005.

[WCK05] Huahui Wu, Mark Claypool, and Robert Kinicki. Adjusting forward error correction with quality scaling for streaming mpeg. In *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 111–116, 2005.

[WMS01] Craig E. Wills, Mikhail Mikhailov, and Hao Shang. N for the price of 1: Bundling web objects for more efficient content delivery. In *Proceedings of the Tenth International World Wide Web Conference*, Hong Kong, May 2001.

[WMS03] Craig E. Wills, Mikhail Mikhailov, and Hao Shang. Inferring relative popularity of Internet applications by actively querying DNS caches. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference*, Miami, Florida, November 2003.

[WS00] Craig E. Wills and Hao Shang. The contribution of DNS lookup costs to web object retrieval. Technical Report WPI-CS-TR-00-12, Worcester Polytechnic Institute, July 2000. `http://www.cs.wpi.edu/~cew/papers/tr00-12.ps.gz`.

[WTM03] Craig E. Wills, Gregory Trott, and Mikhail Mikhailov. Using bundles for web content delivery. *Computer Networks*, 42(6):797–817, August 2003.

[WVSG02] Ren Wang, Massimo Valla, M.Y. Sanadidi, and Mario Gerla. Using adaptive rate estimation to provide enhanced and robust transport over heterogeneous networks. In *Proceedings of the International Conference on Network Protocols*, Paris, France, November 2002. IEEE.

[ZSC91] L. Zhang, S. Shenker, and D.D. Clark. Observations and dynamics of a congestion control algorithm: The effects of two-way traffic". In *Proceedings of ACM SIGCOMM*, pages 133–147, 1991.