On Wide Area Network Optimization

Yan Zhang, *Student Member, IEEE,* Nirwan Ansari, *Fellow, IEEE,* Mingquan Wu, *Member, IEEE,* and Heather Yu, *Member, IEEE,*

***Abstract*—Applications, deployed over a wide area network (WAN) which may connect across metropolitan, regional or national boundaries, suffer performance degradation owing to unavoidable natural characteristics of WANs such as high latency and high packet loss rate. WAN optimization, also known as WAN acceleration, aims to accelerate a broad range of applications and protocols over a WAN. In this paper, we provide a survey on the state of the art of WAN optimization or WAN acceleration techniques, and illustrate how these acceleration techniques can improve application performance, mitigate the impact of latency and loss, and minimize bandwidth consumption. We begin by reviewing the obstacles in efficiently delivering applications over a WAN. Furthermore, we provide a comprehensive survey of the most recent content delivery acceleration techniques in WANs from the networking and optimization point of view. Finally, we discuss major WAN optimization techniques which have been incorporated in widely deployed WAN acceleration products - multiple optimization techniques are leveraged by a single WAN accelerator to improve application performance in general.**

***Index Terms*—Wide area network (WAN), WAN acceleration, WAN optimization, compression, data deduplication, caching, prefetching, protocol optimization.**

I. Introduction

TODAY'S IT organizations tend to deploy their infrastructures geographically over a wide area network (WAN) to increase productivity, support global collaboration and minimize costs, thus constituting to today's WAN-centered environments. As compared to a local area network (LAN), a WAN is a telecommunication network that covers a broad area; WAN may connect across metropolitan, regional, and/or national boundaries. Traditional LAN-oriented infrastructures are insufficient to support global collaboration with high application performance and low costs. Deploying applications over WANs inevitably incurs performance degradation owing to the intrinsic nature of WANs such as high latency and high packet loss rate. As reported in [1], the WAN throughput degrades greatly with the increase of transmission distance and packet loss rate. Given a commonly used maximum window size of 64 KB in the original TCP protocol and 45 Mbps bandwidth, the effective TCP throughput of one flow over a source-to-destination distance of 1000 miles is only around 30% of the total bandwidth. With the source-to-destination distance of 100 miles, the effective TCP throughput degrades from 97% to 32% and 18% of the whole 45 Mbps bandwidth when the packet loss rate increases from 0.1% to 3% and 5%, respectively.

Many factors, not normally encountered in LANs, can quickly lead to performance degradation of applications which are run across a WAN. All of these barriers can be categorized into four classes [2]: network and transport barriers, application and protocol barriers, operating system barriers, and hardware barriers. As compared to LANs, the available bandwidth in WANs is rather limited, which directly affects the application throughput over a WAN. Another obvious barrier in WANs is the high latency introduced by long transmission distance, protocol translation, and network congestions. The high latency in a WAN is a major factor causing the long application response time. Congestion causes packet loss and retransmissions, and leads to erratic behavior of the transport layer protocol, such as transmission control protocol (TCP). Most of the existing protocols are not designed for WAN environments; therefore, several protocols do not perform well under the WAN condition. Furthermore, hosts also impact on the application performance, including operating systems, which host applications; and hardware platforms, which host operating systems.

The need for speedup over WANs spurs on application performance improvement over WANs. The 8-second rule [3] related to a web server's response time specifies that users may not likely wait for a web page if the load-time of the web page exceeds eight seconds. According to an E-commerce web site performance study by Akamai in 2006 [4], this 8-second rule for e-commerce web sites is halved to four seconds, and in its follow-up report in 2009 [5], a new 2-second rule was indicated. These reports showed that poor site performance was ranked second among factors for dissatisfaction and site abandonment. Therefore, there is a dire need to enhance application performance over WANs.

WAN optimization, also commonly referred to as WAN acceleration, describes the idea of enhancing application performance over WANs. WAN acceleration aims to provide high-performance access to remote data such as files and videos. A variety of WAN acceleration techniques have been proposed. Some focus on maximizing bandwidth utilization, others address latency, and still others address protocol inefficiency which hinders the effective delivery of packets across the WAN. The most common techniques, employed by WAN optimization to maximize application performance across the WAN, include compression [6–10], data deduplication [11–24], caching [25–39], prefetching [40–62], and protocol optimization [63–81]. Compression is very important to reduce the amount of bandwidth consumed on a link during transfer across the WAN, and it can also reduce the

Manuscript received 05 May 2011; revised 16 August and 03 September 2011.

Y. Zhang and N. Ansari are with the Advanced Networking Lab., Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, 07102 USA (e-mail: {yz45, nirwan.ansari}@njit.edu).

M. Wu and H. Yu are with Huawei Technologies, USA (e-mail: {Mingquan.Wu, heatheryu}@huawei.com).

Digital Object Identifier 10.1109/SURV.2011.092311.00071

transit time for given data to traverse over the WAN by reducing the amount of transmitted data. Data deduplication is another data reduction technique and a derivative of data compression. It identifies duplicate data elements, such as an entire file and data block, and eliminates both intra-file and inter-file data redundancy, and hence reduces the data to be transferred or stored. Caching is considered to be an effective approach to reduce network traffic and application response time by storing copies of frequently requested content in a local cache, a proxy server cache close to the end user, or even within the Internet. Prefetching (or proactive caching) is aimed at overcoming the limitations of passive caching by proactively and speculatively retrieving a resource into a cache in the anticipation of subsequent demand requests. Several protocols such as common Internet file system (CIFS) [82], (also known as Server Message Block (SMB) [83]), and Messaging Application Programming Interface (MAPI) [84] are chatty in nature, requiring hundreds of control messages for a relatively simple data transfer, because they are not designed for WAN environments. Protocol optimization capitalizes on in-depth protocol knowledge to improve inefficient protocols by making them more tolerant to high latency in the WAN environment. Some other acceleration techniques, such as load balancing, routing optimization, and application proxies, can also improve application performance.

With the dramatic increase of applications developed over WANs, many companies, such as Cisco, Blue Coat, Riverbed Technology, and Silver Peak Systems, have been marketing WAN acceleration products for various applications. In general, typical WAN acceleration products leverage multiple optimization techniques to improve application throughput, mitigate the impact of latency and loss, and minimize bandwidth consumption. For example, Cisco Wide Area Application Services (WAAS) appliance employs data compression, deduplication, TCP optimization, secure sockets layer (SSL) optimization, CIFS acceleration, HyperText Transfer Protocol (HTTP) acceleration, MAPI acceleration, and NFS acceleration techniques to improve application performance. The WAN optimization appliance market was estimated to be $1 billion in 2008 [85]. Gartner, a technology research firm, estimated the compound annual growth rate of the application acceleration market will be 13.1% between 2010 and 2015 [86], and forecasted that the application acceleration market will grow to $5.5 billion in 2015 [86].

Although WAN acceleration techniques have been deployed for several years and there are many WAN acceleration products in the market, many new challenges to content delivery over WANs are emerging as the scale of information data and network sizes is growing rapidly, and many companies have been working on WAN acceleration techniques, such as Google's web acceleration SPDY project [87]. Several WAN acceleration techniques have been implemented in SPDY, such as HTTP header compression, request prioritization, stream multiplexing, and HTTP server push and serve hint. A SPDY-capable web server can respond to both HTTP and SPDY requests efficiently, and at the client side, a modified Google Chrome client can use HTTP or SPDY for web access. The SPDY protocol specification, source code, SPDY proxy examples and their lab tests are detailed in the SPDY web page [87]. As reported in the SPDY tests, up to 64% of page download time reduction can be observed.

WAN acceleration or WAN optimization has been studied for several years, but there does not exist, to the best of our knowledge, a comprehensive survey/tutorial like this one. There is coverage on bits and pieces on certain aspects of WAN optimization such as data compression, which has been widely studied and reported in several books or survey papers, but few works [2, 39] have discussed WAN optimization or WAN acceleration as a whole. Reference [2] emphasizes on application-specific acceleration and content delivery networks while Reference [39] focuses on dynamic web content generation and delivery acceleration techniques. In this paper, we will survey state-of-the-art WAN optimization techniques and illustrate how these acceleration techniques can improve application performance over a WAN, mitigate the impact of latency and loss, and minimize bandwidth consumption. The remainder of the paper is organized as follows. We present the obstacles to content delivery over a WAN in Section II. In order to overcome these challenges in the WAN, many WAN acceleration techniques have been proposed and developed, such as compression, data deduplication, caching, prefetching, and protocol optimization. We detail most commonly used WAN optimization techniques in Section III. Since tremendous efforts have been made in protocol optimization to improve application performance over WAN, we dedicate one section to discuss the protocol optimization techniques over WAN in Section IV, including HTTP optimization, TCP optimization, CIFS optimization, MAPI optimization, session layer optimization, and SSL acceleration. Furthermore, we present some typical WAN acceleration products along with the major WAN optimization techniques incorporated in these acceleration products in Section V; multiple optimization techniques are normally employed by a single WAN accelerator to improve application performance. Finally, Section VI concludes the paper.

II. Obstacles to Content Delivery over a WAN

The performance degradation occurs when applications are deployed over a WAN owing to its unavoidable intrinsic characteristics. The obstacles to content delivery over a WAN can be categorized into four classes [2]: network and transport barriers, application and protocol barriers, operating system barriers, and hardware barriers.

A. Network and Transport Barriers

Network characteristics, such as available bandwidth, latency, packet loss rate and congestion, impact the application performance. Figure 1 summarizes the network and transport barriers to the application performance in a WAN.

1) Limited Bandwidth: The available bandwidth is generally much higher in a LAN environment than that in a WAN environment, thus creating a bandwidth disparity between these two dramatically different networks. The limited bandwidth impacts the capability of an application to provide high throughput. Furthermore, oversubscription or aggregation is generally higher in a WAN than that in a LAN. Therefore, even though the clients and servers may connect to the edge routers

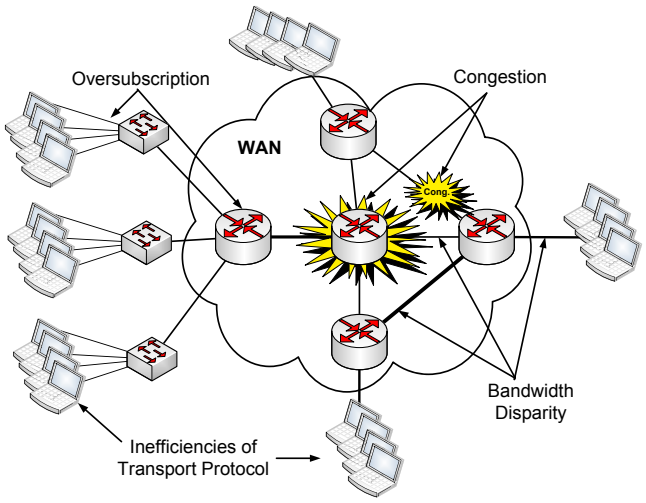


Fig. 1. Network and transport barriers to application performance over a WAN.

with high-speed links, the overall application performance over a WAN is throttled by network oversubscription and bandwidth disparity because only a small number of requests can be received by the server, and the server can only transmit a small amount of data at a time in responding to the clients' requests. Protocol overhead, such as packet header and acknowledgement packets, consumes a noticeable amount of network capacity, hence further compromising the application performance.

2) High Latency: The latency introduced by transmission distance, protocol translation, and congestion is high in the WAN environment, and high latency is the major cause for long application response time over a WAN.

3) Congestion and High Packet Loss Rate: Congestion causes packet loss and retransmission, and leads to erratic behaviors of transport layer protocols that may seriously deteriorate the application performance.

B. Application and Protocol Barriers

The application performance is constantly impacted by the limitations and barriers of the protocols, which are not designed for WAN environments in general. Many protocols do not perform well under WAN conditions such as long transmission path, high network latency, network congestion, and limited available bandwidth. Several protocols such as CIFS and MAPI are chatty in nature, requiring hundreds of control messages for a relatively simple data transfer. Some other popular protocols, e.g., Hypertext Transfer Protocol (HTTP) and TCP, also experience low efficiency over a WAN. A detailed discussion on protocol barriers and optimizations in a WAN will be presented in Section IV.

C. Operating System and Hardware Barriers

The hosts, including their operating systems, which host the applications; and hardware platforms, which host operating systems, also impact the application performance. Proper selection of the application hosts' hardware and operating system components, including central processing unit, cache capacity, disk storage, and file system, can improve the overall application performance. A poorly tuned application server will have a negative effect on the application's performance and functionality across the WAN. In this survey, we focus on the networking and protocol impacts on the application performance over WAN. A detailed discussion on the performance barriers caused by operating systems and hardware platforms, and guidance as to what aspects of the system should be examined for a better level of application performance can be found in [2].

III. WAN Optimization Techniques

WAN acceleration technologies aim, over a WAN, to accelerate a broad range of applications and protocols, mitigate the impact of latency and loss, and minimize bandwidth consumption. The most common techniques employed by WAN optimization to maximize application performance across the WAN include compression, data deduplication, caching, prefetching, and protocol optimization. We discuss the following most commonly used WAN optimization techniques.

A. Compression

Compression is very important to minimize the amount of bandwidth consumed on a link during transfer across the WAN in which bandwidth is quite limited. It can improve bandwidth utilization efficiency, thereby reducing bandwidth congestion; it can also reduce the amount of transit time for given data to traverse the WAN by reducing the transmitted data. Therefore, compression substantially optimizes data transmission over the network. A comparative study between various text file compression techniques is reported in [6]. A survey on XML compression is presented in [7]. Another survey on lossless image compression methods is presented in [8]. A survey on image and video compression is covered in [9].

HTTP [88, 89] is the most popular application-layer protocol in the Internet. HTTP compression is very important to enhance the performance of HTTP applications. HTTP compression techniques can be categorized into two schemes: HTTP Protocol Aware Compression (HPAC) and HTTP Bi-Stream Compression (HBSC) schemes. By exploiting the characteristics of the HTTP protocol, HPAC jointly uses three different encoding schemes, namely, Stationary Binary Encoding (SBE), Dynamic Binary Encoding (DBE), and Header Delta Encoding (HDE), to perform compression. SBE can compress a significant amount of ASCII text present in the message, including all header segments except request-URI (Uniform Resource Identifier) and header field values, into a few bytes. The compressed information is static, and does not need to be exchanged between the compressor and decompressor. All those segments of the HTTP header that cannot be compressed by SBE will be compressed by DBE. HDE is developed based on the observation that HTTP headers do not change much for an HTTP transaction and a response message does not change much from a server to a client. Hence, tremendous information can be compressed by sending only the changes of a new header from a reference. HBSC is an

TABLE I
COMPRESSION SAVINGS FOR DIFFERENT WEB SITE CATEGORIES [90]

Web Site Type	Text File Only	Overall (Graphics)
High-Tech Company	79%	35%
Newspaper	79%	40%
Web Directory	69%	46%
Sports	74%	27%
Average	75%	37%

algorithm-agnostic framework that supports any compression algorithms. HBSC maintains two independent contexts for the HTTP header and HTTP body of a TCP connection in each direction to avoid the problem of context thrashing. These two independent contexts are created when the first message appears for a new TCP connection, and are deleted until this TCP connection finishes; thus, the inter-message redundancy can be detected and removed since the same context is used to compress HTTP messages at one TCP connection. HBSC also pre-populates the compression context for HTTP headers with text strings in the first message over a TCP connection, and detects the compressibility of an HTTP body based on the information in the HTTP header to further improve the compression performance. HTTP header compression has been implemented in Google's SPDY project [87] - according to their results, HTTP header compression resulted in an about 88% reduction in the size of HTTP request headers and an about 85% reduction in the size of HTTP response headers. A detailed description of a set of methods developed for HTTP compression and their test results can be found in [10].

The compression performance for web applications depends on the mix of traffic in the WAN such as text files, video and images. According to Reference [90], compression can save 75 percent of the text file content and save 37 percent of the overall file content including graphics. Their performance of compression is investigated based on four different web site categories, including high technology companies, newspaper web sites, web directories, and sports. For each category, 5 web sites are examined. Table I lists the precentage of bytes savings after compression for different investigated web site categories.

B. Data Deduplication

Data deduplication, also called redundancy elimination [11, 12], is another data reduction technique and a derivative of data compression. Data compression reduces the file size by eliminating redundant data contained in a document, while data deduplication identifies duplicate data elements, such as an entire file [13, 14] and data block [15–23], and eliminates both intra-file and inter-file data redundancy, hence reducing the data to be transferred or stored. When multiple instances of the same data element are detected, only one single copy of the data element is transferred or stored. The redundant data element is replaced with a reference or pointer to the unique data copy. Based on the algorithm granularity, data deduplication algorithms can be classified into three categories: whole file hashing [13, 14], sub-file hashing [15–23], and delta encoding [24]. Traditional data de-duplication operates at the application layer, such as object caching, to eliminate redundant data transfers. With the rapid growth of network traffic in the Internet, data redundancy elimination techniques operating on individual packets have been deployed recently [15–20] based on different chunking and sampling methods. The main idea of packet-level redundancy elimination is to identify and eliminate redundant chunks across packets. A large scale trace-driven study on the efficiency of packet-level redundancy elimination has been reported in [91]. This study showed that packet-level redundancy elimination techniques can obtain average bandwidth savings of 15-60% when deployed at access links of the service providers or between routers. Experimental evaluations on various data redundancy elimination technologies are presented in [11, 18, 91].

C. Caching

Caching is considered to be an effective approach to reduce network traffic and application response time. Based on the location of caches, they can be deployed at the client side, proxy side, and server side. Owing to the limited capacity of a single cache, caches can also work cooperatively to serve a large number of clients. Cooperative caching can be set up hierarchically, distributively, or in a hybrid mode. From the type of the cached objects, caches can be classified as function caching and content caching. A hierarchical classification of caching solutions is shown in Figure 2.

1) Location of cache: Client side caches are placed very close to and even at the clients. All the popular web browsers, including Microsoft Internet Explorer and Mozilla Firefox, use part of the storage space on client computers to keep records of recently accessed web content for later reference to reduce the bandwidth used for web traffic and user perceived latency.

Several solutions [25–27] have been proposed to employ client cache cooperation to improve client-side caching efficiency. Squirrel [25], a decentralized, peer-to-peer web cache, was proposed to enable web browsers on client computers to share their local caches to form an efficient and scalable web cache. In Squirrel, each participating node runs an instance of Squirrel, and thus web browsers will issue their requests to the Squirrel proxy running on the same machine. If the requested object is un-cacheable, the request will be forwarded to the origin server directly. Otherwise, the Squirrel proxy will check the local cache. If the local cache does not have the requested object, Squirrel will forward the request to some other node in the network. Squirrel uses a self-organizing peer-to-peer routing algorithm, called Pastry, to map the requested object URL as a key to a node in the network to which the request will be forwarded. One drawback of this approach is that it neglects the diverse availabilities and capabilities among client machines. The whole system performance might be affected by some low capacity intermediate nodes since it takes several hops before an object request is served. Figure 3 illustrates an example of the Squirrel requesting and responding procedure. Client s issues a request to client j with 2 hops routing through

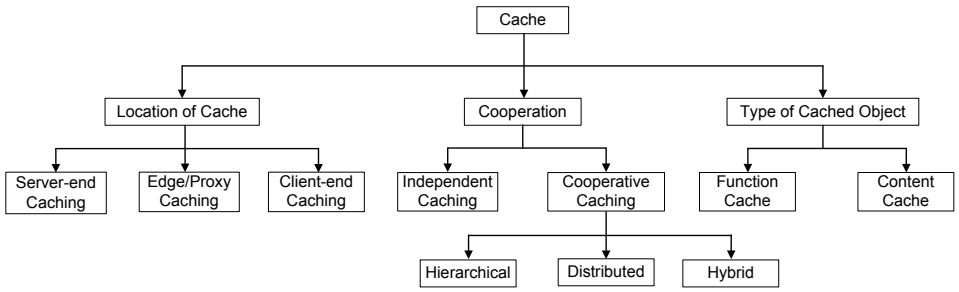


Fig. 2. Caching classification hierarchy.

client i. If the requested object is present in the browser cache of client j, the requested object will be forwarded back to client s directly through path A. Otherwise, client j will forward the request to the origin server, and the origin server will respond the request through path B.

Xiao *et al.* [26] proposed a peer-to-peer Web document sharing technique, called browsers-aware proxy server, which connects to a group of networked clients and maintains a browser index file of objects contained in all client browser caches. A simple illustration of the organization of a browsers-aware proxy server is shown in Figure 4. If a cache miss in its local browser cache occurs, a request will be generated to the proxy server, and the browsers-aware proxy server will check its proxy cache first. If it is not present in the proxy cache, the proxy server will look up the browser index file attempting to find it in other client's browser cache. If such a hit is found in a client, this client will forward the requested object directly to the requesting client; otherwise, the proxy server will send the request to an upper level proxy or the origin server. The browser-aware proxy server suffers from the scalability issue since all the clients are connected to the centralized proxy server.

Xu *et al.* [27] proposed a cooperative hierarchical client-cache technique. A large virtual cache is generated from contribution of local caches of each client. Based on the client capability, the clients are divided into the super-clients and the ordinary clients, and the system workload will be distributed among these relatively high capacity super-clients. Unlike the browsers-aware proxy server scheme, the super-clients are only responsible for maintaining the location information of the cached files; this provides high scalability because caching and data lookup operations are distributed across all clients and super-clients. Hence, such a hierarchical web caching structure reduces the workloads on the dedicated server in browsers-aware proxy server scheme [26] and also relieves the weak client problem in Squirrel [25].

Contrary to the client-side caching, server-end caches are placed very close to the origin servers. Server-end caching can reduce server load and improve response time especially when the client stress is high. For edge/proxy side caching [28], caches are placed between the client and the server. According to the results reported in [29], local proxy caching could reduce user perceived latency by at best 26%.

2) Cooperative Caching: Owing to the limited capacity of single caches, multiple caches can share and coordinate the cache states to build a cache network serving a large number of users. Cooperative caching architectures can be classified into three major categories [30]: hierarchical cooperative caching [31], distributive cooperative caching [32–37], and hybrid cooperative caching [38].

In the hierarchical caching architecture, caches can be placed at different network levels, including client, institutional, regional and national level from bottom to top in the hierarchy. Therefore, it is consistent with present Internet architecture. If a request cannot be satisfied by lower level caches, it will be redirected to upper level caches. If it cannot be served by any cache level, the national cache will contact the origin server directly. When the content is found, it travels down the hierarchy, leaving a copy at each of the intermediate caches. One obvious drawback of the hierarchical caching system is that multiple copies of the same document are stored at different cache levels. Each cache level introduces additional delays, thus yielding poor response times. Higher level caches may also experience congestion and have long queuing delays to serve a large number of requests.

In distributed caching systems, there are only institutional caches at the edge of the network. The distributed caching system requires some mechanisms to cooperate these institutional caches to serve each other's cache misses. Several mechanisms have been proposed so far, including the query-based approach, content list based approach, and hash function based approach. Each of them has its own drawbacks. A query-based approach such as Inter Cache Protocol (ICP) [32] can be used to retrieve the document which is not present at the local cache from other institutional caches. However, this method may increase the bandwidth consumption and the user perceived latency because a cache have to poll all cooperating caches and wait for all of them to answer. A content list of each institutional cache, such as cache digest [33] and summary cache [36], can help to avoid the need for queries/polls. In order to distribute content lists more efficiently and scalably, a hierarchical infrastructure of immediate nodes is set up in general, but this infrastructure does not store any document copies. A hash function [35] can be used to map a client request into a certain cache, and so there is only one single copy of a document among all cooperative caches; this method is thus limited to the local environment with well-interconnected caches.

Rodriguez *et al.* [30] proposed analytical models to study and compare the performance of both hierarchical and distributed caching. The derived models can be used to calculate the user perceived latency, the bandwidth utilization, the disk

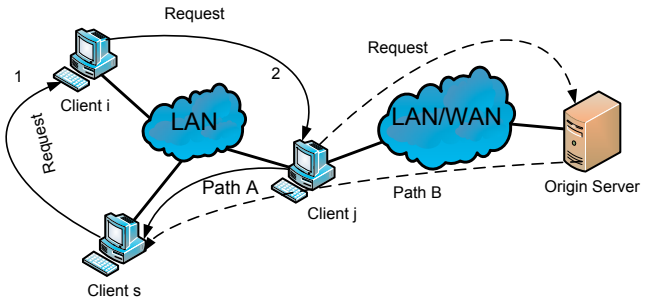


Fig. 3. A Simple illustration of Squirrel requesting and responding procedure. The request is handled in one of two possible ways, path A or path B [25].

space requirements, and the load generated by each cache cooperating scheme. In order to maximize the advantages and minimize the weaknesses of both hierarchical and distributed caching architectures, hybrid caching architecture has been proposed [30, 38]. In a hybrid caching scheme, the cooperation among caches may be limited to the same level or at a higher level caches only. Rabinovich *et al.* [38] proposed a hybrid caching scheme, in which the cooperation is limited between the neighboring caches to avoid obtaining documents from distant or slower caches. The performance of the hybrid scheme and the optimal number of caches that should cooperate at each caching level to minimize user retrieval latency has been investigated in [30].

3) Type of Cached Objects: Based on the type of the cached objects, caches can be categorized as content caching and function caching. In content caching, objects such as documents, HTML pages, and videos are stored. Content caching is employed widely to reduce the bandwidth consumed for traffic traveling across the network and user perceived latency. In function caching, the application itself is replicated and cached along with its associated objects, and so the proxy servers can run applications instead of the origin server. A detailed review on content caching and function caching for dynamic web delivery is reported in [39].

D. Prefetching

Although caching offers benefits to improve application performance across the WAN, passive caching has limitations on reducing application latency due to low hit rates [29, 40]. Abrams *et al.* [40] examined the hit rates for various workloads to investigate the removal policies for caching within the Web. Kroeger *et al.* [29] confirmed a similar observation that local proxy caching could reduce latency by at best 26% under several scenarios. They also found that the benefit of caching is limited by the frequency update of objects in the web. Prefetching (or proactive caching) is aimed at overcoming the limitations of passive caching by proactively speculative retrieval of a resource into a cache in the anticipation of subsequent demand requests. The experiments reported in [29] showed that prefetching doubles the latency reduction achieved by caching, and a combination of caching and prefetching can provide a better solution to reduce latency than caching and prefetching alone. So far, several web prefetching architectures have been proposed according to the locations of the prediction engine and prefetching engine, which are two main elements of web prefetching architecture. In addition, many prediction algorithms have been proposed, and how far in advance a prefetching algorithm is able to prefetch an object is a significant factor in its ability to reduce latency. The effectiveness of prefetching in addressing the limitations of passive caching has been demonstrated by several studies. Most research on prefetching focused on the prediction algorithm. The prediction algorithms used for prefetching systems can be classified into two categories: history-based prediction [43, 44, 50, 52–57] and content-based prediction [58–62].

1) Web Prefetching Architecture: In general, clients and web servers form the two main elements in web browsing, and optionally, there may be proxies between the clients and servers. A browser runs at the client side and a HTTP daemon runs at the server side. A web prefetching architecture consists of a prediction engine and a prefetching engine. The prediction engine in the web prefetching architecture will predict the objects that a client might access in the near future, and the prefetching engine will preprocess those object requests predicted by the prediction engine. Both the prediction engine and prefetching engine can be located at the client, at the proxy, or at the server. Several different web prefetching architectures have been developed.

(i) Client-initiated prefetching. Both the prediction engine and prefetching engine can be located at the clients [47] to reduce the interaction between the prediction engine and prefetching engine.

(ii) Server-initiated prefetching. Server can predict clients' future access, and clients perform prefetching [43–46].

(iii) Proxy-initiated prefetching (Proxy Predicting and Proxy Pushing or Client Prefetching). If there are proxies in a web prefetching system, the prediction engine can be located at the proxies and the predicted web objects can be pushed to the clients by the proxies, or the clients can prefetch in advance [50].

(iv) Collaborative prefetching. The prediction accuracy of proxy-based prefetching can be significantly limited without input of Web servers. A coordinated proxy-server prefetching can adaptively utilize the reference information and coordinate prefetching activities at both proxy and web servers [48]. As reported in [41], a collaborative client-server prediction can also reduce the user's perceived latency.

(v) Multiple level prefetching. A web prefetching architecture with two levels caching at both the LAN proxy server and the edge router connecting to the Internet was proposed in [49] for wide area networks. The edge router is responsible for request predictions and prefetching. In [55], the prefetching occurs at both clients and different levels of proxies, and servers collaborate to predict users' requests [55].

The impact of the web architecture on the limits of latency reduction through prefetching has been investigated in [41]. Based on the assumption of realistic prediction, the maximum latency reduction that can be achieved through prefetching is about 36.6%, 54.4% and 67.4% of the latency perceived by users with the prefetching predictor located

at the server, client, and proxy, respectively. Collaborative prediction schemes located at diverse elements of the web architecture were also analyzed in [41]. At the maximum, more than 95% of the user's perceived latency can be achieved with a collaborative client-server or proxy-server predictor. A detailed performance analysis on the client-side caching and prefetching system is presented in [42].

2) History Based Prediction Algorithms: The history based prediction algorithms relies on the user's previous actions such as the sequence of requested web objects to determine the next most likely ones to be requested. The major component of a history-based prediction algorithm is the Prediction by Partial Matching (PPM) model [92], which is derived from data compression. PPM compressor uses the preceding few characters to compute the probability of the next ones by using multiple high-order Markov models. A standard PPM prediction model has been used by several works for web prefetching [43, 44, 50]. Following the standard PPM model, several other PPM based prediction models, such as LRS-PPM model [51] and popularity-based PPM model [52], have been proposed.

Based on the preceding m requests by the client over some period, the standard PPM model essentially tries to make predictions for the next l requests with a probability threshold limit t. Larger m, meaning more preceding requests to construct the Markov predictor tree, can improve the accuracy of the prediction. With $l > 1$, URLs within the next few requests can be predicted. A server initiated prefetching scheme with a dependency graph based predictor was proposed in [43]. The dependency graph contains nodes for all files ever accessed at a particular web server. The dependency arc between two nodes indicates the probability of one file, represented as a node, will be accessed after the other one is accessed. The dependency graph based prediction algorithm is a simplified first-order standard PPM model with $m = 1$, indicating that the predicted request is only based on the immediate previous one. First-order Markov models are not very accurate in predicting the users browsing behavior because these models do not look far into the past to correctly discriminate the different observed patterns. Therefore, as an extension, Palpanas [44] investigated the effects of several parameters on prediction accuracy in a server-initiated prefetching with a standard PPM prediction engine. The impacts of previous m requests and the number of predictions l on prediction accuracy have been examined, and the experimental results reported in [44] showed that more previous requests m result in more accurate predictions, while more predicted requests l with the same previous requests m degrade prediction accuracy. A proxy-initiated prefetching with the standard PPM model was proposed in [50], and from their results, it was shown that the best performance is achieved with $m = 2$ and $l = 4$.

High-order Markov model can improve the prediction accuracy, but unfortunately, these higher-order models have a number of limitations associated with high state-space complexity, reduced coverage, and sometimes even worse prediction accuracy. Sarukkai [53] used Markov models to predict the next page accessed by the user by analyzing users' navigation history. Three selective Markov models, support-

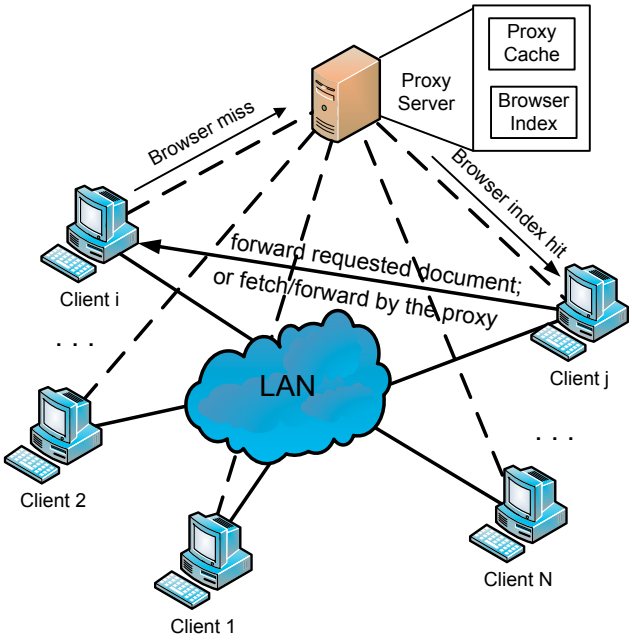


Fig. 4. Organizations of the browser-aware proxy server [26].

pruned Markov model (SPMM), confidence-pruned Markov model (CPMM), and error-pruned Markov model (EPMM), have been proposed in [54] to reduce state complexity of high order Markov models and improve prediction accuracy by eliminating some states in Markov models with low prediction accuracy. SPMM is based on the observation that states supported by fewer instances in the training set tend to also have low prediction accuracy. SPMM eliminates all the states, the number of instances of which in the training set is smaller than the frequency threshold, in different order Markov models. CPMM eliminates all the states in which the probability differences between the most frequently taken action and the other actions are not significant. EPMM uses error rates, which are determined from a validation step, for pruning.

Markatos *et.al* [55] suggested a Top-10 criterion for web prefetching. The Top-10 criterion based prefetching is a client-server collaborative prefetching. The server is responsible for periodically calculating its top 10 most popular documents, and serve them to its clients. Chen *et al.* [52] proposed a popularity-based PPM prediction model which uses grades to rank URL access patterns and builds these patterns into a predictor tree to aid web prefetching.

Most of the above prediction algorithms employ Markov models to make decisions. Some other prediction algorithms rely on data mining techniques to predict a new comer's request. A web mining method was investigated in [51] to extract significant URL patterns by the longest repeating subsequences (LRS) technique. Thus, the complexity of the prediction model can be reduced by keeping the LRS and storing only long branches with frequently accessed URLs. Nanopoulos *et al.* [56] proposed a data mining based prediction algorithm called WM_o (o stands for ordering), which improves the web prefetching performance by addressing some specific limitations of Markov model based prediction

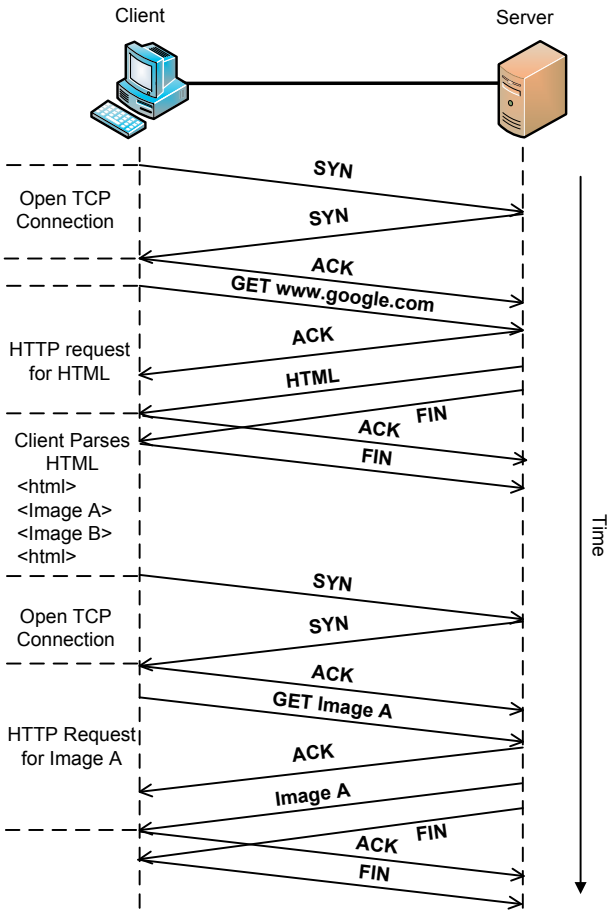


Fig. 5. The standard packet exchange for HTTP 1.0 [88].

algorithms, such as the order of dependencies between web document accesses, and the ordering of requests. Songwattana [57] proposed a prediction-based web caching model, which applies web mining and machine learning techniques for prediction in prefetching and caching. In this prediction-based model, statistical values of each web object from empirical data test set are generated by ARS log analyzer based on the web log files, and a web mining technique is used to identify the web usage patterns of clients requests.

3) Content Based Prediction Algorithms: A limitation of history based prediction is that the predicted URLs must be accessed before and there is no way to predict documents that are new or never accessed. To overcome this limitation, several content-based prediction algorithms [58–62] have been proposed. The content-based prediction algorithms can extract top ranked links by analyzing the content of current page and previously viewed pages. Dean and Henzinger [58] proposed a web searching algorithm by using only the connectivity information in the web between pages to generate a set of related web pages. In a web prefetching scheme proposed in [59], users send usage reports to servers promptly. The usage report of one particular page includes the information about the URLs embedded within this page that is referenced recently, the size of the referenced page, and all its embedded images. The server will accumulate the information from all usage reports on this particular page to generate a usage profile of this page, and send it to the clients. The clients will make prefetching decisions based on its own usage patterns, the state of its cache, and the effective bandwidth of its link to the Internet. The prefetching by combining the analysis of these usage profiles can lower the user perceived latency and reduce the wastage of prefetched pages. Ibrahim and Xu [60] proposed a keyword-oriented neural network based prefetching approach, called SmartNewsReader, which ranks the links of a page by a score computed by applying neural networks to the keywords extracted from the URL anchor texts of the clicked links to characterize user access patterns. Predictions in [61] are made on the basis of the content of the recently requested Web pages. Georgakis and Li [62] proposed a transparent and speculative algorithm for content based web page prefetching, which relies on the user profile on the basis of a user's Internet browsing habits. The user profile describes the frequency of occurrences of selected elements in a web page clicked by the user. These frequencies are used to predict the user's future action.

4) Prefetching Performance Evaluation: It is difficult to compare the proposed prefetching techniques since different baseline systems, workload, and performance key metrics are used to evaluate their benefits. In general, prefetching performance can be evaluated from three aspects. The first one is the prediction algorithm, including prediction accuracy and efficiency. The additional resource usage that prefetching incurs is another evaluation criterion. The prefetching performance can also been evaluated by users' perceived latency reduction. A detailed discussion on prefetching performance metrics can be found in [93].

E. Other Techniques

Load balancing offers resilience and scalability by distributing data across a computer network to maximize network efficiency, and thus to improve overall performance. Route Optimization is the process of choosing the most efficient path by analyzing performance between routes and network devices, and sending data in that path. This enables users to achieve faster application response and data delivery. Transparent or nontransparent application proxies are primarily used to overcome application latency. By understanding application messaging, application proxies can handle the unnecessary messages locally or by batching them for parallel operation; they can also act in advance, such as *read ahead* and *write behind*, to reduce the application latency. As discussed in Section II, packet loss rate in WANs is high. When high packet loss rate is coupled with high latency, it is not surprising that application performance suffers across a WAN. Forward Error Correction (FEC) corrects bit errors at the physical layer. This technology is often tailored to operate on packets at the network layer to mitigate packet loss across WANs that have high packet loss characteristics. Packet Order Correction (POC) is a real-time solution for overcoming out-of-order packet delivery across the WAN.

IV. Protocol Optimization

Protocol optimization is the use of in-depth protocol knowledge to improve inefficient protocols by making them more

tolerant to high latency in WAN environments. Several protocols such as CIFS and MAPI are chatty in nature, requiring hundreds of control messages for a relatively simple data transfer, because they are not designed for WAN environments. Some other popular protocols, e.g., HTTP and TCP, also experience low efficiency over a WAN. In this section, we will discuss HTTP optimization, TCP acceleration, CIFS optimization, MAPI optimization, session layer acceleration, and SSL/TLS acceleration.

A. HTTP Optimization

Users always try to avoid slow web sites and flock towards fast ones. A recent study found that users expect a page to load in two seconds or less, and 40% of users will wait for no more than three seconds before leaving a site [94]. HTTP is the most popular application layer protocol used by web applications. Currently, there are two versions of HTTP protocols: HTTP 1.0 [88] and HTTP 1.1 [89]. TCP is the dominant transport layer protocol in use for HTTP applications; it adds significant and unnecessary overhead and increases user perceived latency. Many efforts have been made to investigate HTTP performance, reduce the bandwidth consumption, and accelerate the web page transmission. Persistent connections, pipeline and compression introduced in HTTP 1.1 are the basic HTTP acceleration techniques. HTTP compression techniques have been discussed in Section III-A. Caching and prefetching, which have been presented in Section III-C and III-D, respectively, help to reduce user perceived latency. L4-7 load balancing also contributes to HTTP acceleration. HTTP accelerators are offered to the market by Squid, Blue Coat, Varnish, Apache, etc.

1) Inefficiency of HTTP Protocol Analysis: Figures 5 and 6 illustrate the standard packet exchange procedures when a client visits a web site with HTTP 1.0 and HTTP 1.1, respectively. As shown in these two figures, an URL request, which executes the HTTP protocol, begins with a separate TCP connection setup and follows by a HyperText Markup Language (HTML) file transiting from the server to the client. Since embedded objects are referenced with the HTML file, these files cannot be requested until at least part of the HTML file has been received by the client. Therefore, after the client receives the HTML file, it parses the HTML file and generates object requests to retrieve them from the server to the client. In HTTP 1.0, each HTTP request to retrieve an embedded object from the server has to set up its own TCP connection between the client and the server, thus placing additional resource constraint on the server and client than what would be experienced using a single TCP connection. Hence, as depicted in Figure 5, at least 4 RTTs are required before the first embedded object is received at the client. The low throughput of using this per-transaction TCP connection for HTTP access has been discussed in [63]. The reduced throughput increases the latency for document retrieval from the server. To improve the user perceived latency in HTTP 1.0, persistent connections and request pipelining are introduced in HTTP 1.1.

2) Persistent Connection and Request Pipelining: Persistent connection, using a single, long-lived connection for multiple HTTP transactions, allows the TCP connection stays open after each HTTP request/response transaction, and thus the next request/response transaction can be executed immediately without opening a new TCP connection as shown in Figure 6. Hence, persistent connections can reduce server and client workload, and file retrieval latency. Even with persistent connections, at least one round-trip time is required to retrieve each embedded object in the HTML file. The client interacts with the server still in a stop-and-wait fashion, sending a HTTP request for an embedded object only after having received the previous requested object. Request pipelining allows multiple HTTP requests sent out to the same server to retrieve the embedded objects. The effects of persistent connections and request pipelining on the HTTP exchange latency have been investigated in [63]. The results show that persistent connections and request pipelining indeed provide significant improvement for HTTP transactions. The user perceived latency improvement decreases with the increase of the requested document size since the actual data transfer time dominates the connection setup and slow-start latencies to retrieve large documents.

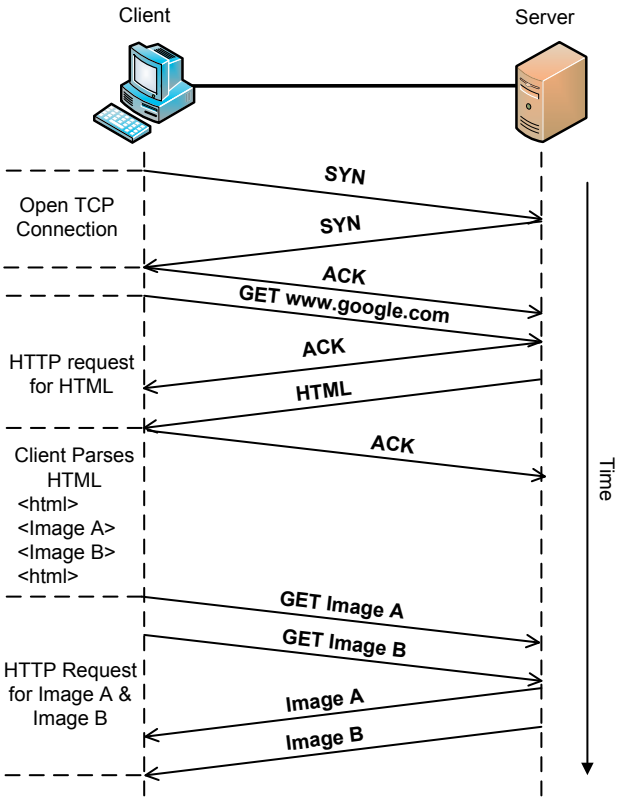


Fig. 6. The standard packet exchange for HTTP 1.1 with persistent connection and request pipelining [89].

3) HTML Plus Embedded Objects: In HTTP 1.0 and 1.1, one common procedure is that the client has to wait for the HTML file before it can issue the requests for embedded objects, thus incurring additional delay to user perceived latency. In order to avoid this additional delay, Abhari and Serbinski [64] suggested a modification to the HTTP protocol. Before responding to the client with the requested HTML file, the HTTP server parses the requested HTML documents

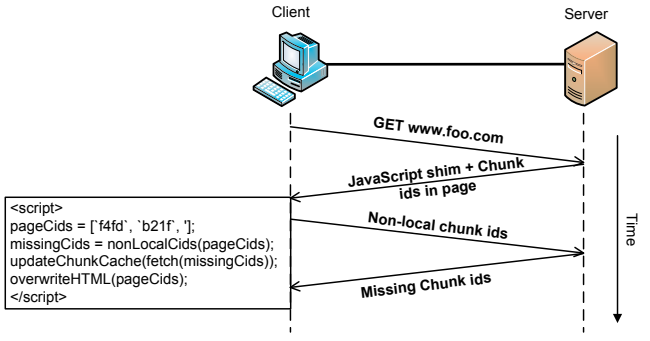
to determine which objects are embedded in this requested HTML file and appends them onto the response to the client. Since a normal HTTP client does not have the ability to receive multiple files in a single transmission, a proxy is suggested to be placed at the client side to translate client request and server response. The experiments reported in [64] show a significant page load time reduction of the proposed scheme.

4) Minimizing HTTP Requests: The web page load time can be decreased by reducing the number of HTTP requests. The concatenated JavaScript files and Cascading style sheets (CSS) files through object inlining can reduce the number of HTTP requests. Object inlining can reduce the page load latency significantly, but the browser can only cache URL addressable objects; the individual HTML, CSS and JavaScript files cannot be cached. Furthermore, if any of the embedded objects change, the browsers have to retrieve all of the objects.

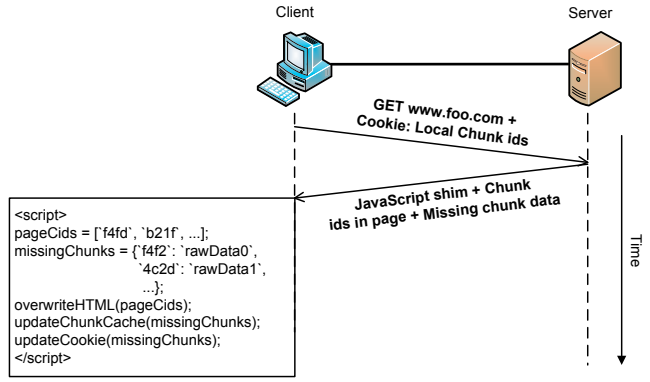
Mickens [65] developed a new framework for deploying fast-loading web applications, called Silo, by leveraging on JavaScript and document object model (DOM) storage to reduce both the number of HTTP requests and the bandwidth required to construct a page. A basic web page fetching procedure using the Silo protocol is depicted in Figure 7(a). Instead of responding with the page's HTML to a standard GET request for a web page, the Silo enabled server replies with a small piece of JavaScript, called JavaScript shim, which enables the client to participate in the Silo protocol, and a list of chunk ids in the page to be constructed. The JavaScript shim will inspect the client DOM storage to determine which of these chunks are not stored locally, and inform the missing chunk ids to the server. Then, the server will send the raw data for the missing chunks. The client assembles the relevant chunks to overwrite the page's HTML and reconstructs the original inlined page. Thus, the basic Silo protocol fetches an arbitrary number of HTML, CSS and JavaScript files with two RTTs.

The basis Silo protocol enabled server does not differentiate between clients with warm caches and clients with cold caches. Soli uses cookies to differentiate clients by setting a "warm cache" variable in the page's cookie whenever clients store chunks for that page. If this variable is set when the client initiates HTTP GET request for a web page, the server knows that the client chunk cache is warm, and otherwise it is cold. Figure 7(b) shows a single RTT Silo protocol with warm client cache. The client initiates an HTTP GET request for a page with a warm cache indicator. If the client attaches all of the ids of the local chunks within the cookie, the server can reply with the Silo shim and the missing chunk data in a single server response; otherwise, the basic Soli protocol depicted in Figure 7(a) will follow the client-server exchange process. As compared to the server response to request with the warm cache, the server replies with an annotated HTML to a standard GET request for a page with a cold cache indication. The browser commits the annotated HTML immediately, while the Silo shim parses the chunk manifest, extracts the associated chunks, and writes them to the DOM storage synchronously.

5) HTTP over UDP: Cidon *et al.* [66] proposed a hybrid TCP-UDP transport for HTTP traffic that splits the HTTP web traffic between UDP and TCP. A client issues a HTTP request over UDP. The server replies over UDP if the response is small; otherwise, the server informs the client to resubmit request over TCP. If no response is received within a timeout, the client resubmits the response over TCP.



(a) The basic Silo protocol.



(b) Single RTT Silo protocol, warm client cache.

Fig. 7. Silo HTTP protocol [65].

Dual-Transport HTTP (DHTTP) [67], a new transfer protocol for the web, was proposed by splitting the traffic between UDP and TCP channels based on the size of server response and network conditions. In DHTTP, there are two communication channels between a client and a server, a UDP channel and a TCP channel. The client usually issues its requests over the UDP channel. The server responds to a request of 1460 bytes or less over the UDP channel, while the response to other requests will be transmitted over the TCP channel. The performance analysis performed in [67] shows that DHTTP significantly improves the performance of Web servers, reduces user latency, and increases utilization of remaining TCP connections. However, DHTTP imposes extra requirements on firewalls and network address translation (NAT) devices.

6) HTTP Packet Reassembly: HTTP packet reassembly is quite normal in current networks. As examined in [68], the HTTP packets reassembly proportions are about 70%-80%, 80%-90%, and 60%-70% for web application HTTP OK messages, HTTP POST messages, and video streaming packets, respectively. Since HTTP does not provide any reassembly mechanism, HTTP packet reassembly has to be completed in the IP layer and TCP layer. Traditionally, HTTP packets reassembly consist of network packet capture, IP layer defragmentation, and TCP layer reassembly before it is passed to the HTTP protocol. A parallel HTTP packet reassembly

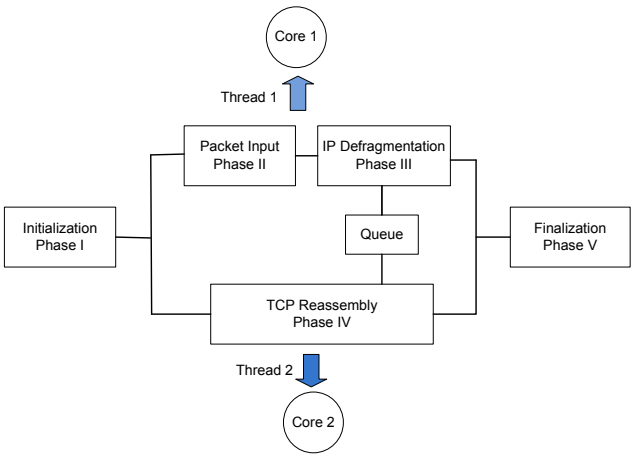


Fig. 8. Parallel HTTP packet reassembly architecture [68].

strategy was proposed in [68] as shown in Figure 8. The two most resource-consuming parts, IP layer defragmentation and TCP layer reassembly, are distributed to two separate threads to facilitate parallelism. Since the workload of IP defragmentation is lower than that of TCP reassembly in the current network, the packet input module is distributed to the same thread with IP defragmentation. IP layer defragmentation and TCP layer reassembly communicate with a single-producer single-consumer ring queue. Hence, different tasks in HTTP packet reassembly can be distributed to different core with a multi-core server using the affinity-based scheduling. According to their experiments, the throughput of packets reassembly system can be improved by around 45% on the generic multi-core platform by parallelizing the traditional serial HTTP packets reassembly system.

B. TCP Acceleration

Currently, TCP is the most popular transport layer protocol for connection-oriented reliable data transfer in the Internet. TCP is the de facto standard for Internet-based commercial communication networks. The growth trends of TCP connections in WANs were investigated in [95]. However, TCP is well known to have poor performance under conditions of moderate to high packet loss and end-to-end latency.

1) Inefficiency of TCP over WAN: The inefficiency of the TCP protocol over WAN comes from two mechanisms, slow start and congestion control. Slow start helps TCP to probe the available bandwidth. Many application transactions work with short-live TCP connections. At the beginning of a connection, TCP is trying to probe the available bandwidth while the application data is waiting to be transmitted. Thus, TCP slow start increases the number of round trips, thus delaying the entire application and resulting in inefficient capacity utilization.

The congestion avoidance mechanism adapts TCP to network dynamics, including packet loss, congestion, and variance in available bandwidth, but it degrades application performance greatly in WAN where the round-trip time is generally in tens even hundreds of milliseconds. That is, it can take quite some time for the sender to receive an acknowledgement indicating to TCP that the connection could increase its throughput slightly. Coupled with the rate at which TCP increases throughput, it could take hours to return to the maximum link capacity over a WAN.

2) TCP Models over WAN: Besson [96] proposed a fluid-based model to describe the behavior of a TCP connection over a WAN, and showed that two successive bottlenecks seriously impact the performance of TCP, even during congestion avoidance. The author also derived analytical expressions for the maximum window size, throughput, and round-trip time of a TCP connection. Mirza *et al.* [97] proposed a machine learning approach to predict TCP throughput, which was implemented in a tool called *PathPerf*. The test results showed that *PathPerf* can predict TCP throughput accurately over diverse wide area paths. A systematic experimental study of IP transport technologies, including TCP and UDP, over 10 Gbps wide-area connections was performed in [98]. The experiments verified the low TCP throughput in the WAN due to high network latency, and the experimental results also showed that the encryption devices have positive effects on TCP throughput due to their on-board buffers.

3) Slow Start Enhancement: Many studies have observed that TCP performance suffers from the TCP slow start mechanism in high-speed long-delay networks. TCP slow start can be enhanced from two aspects by setting *ssthresh* intelligently and adjusting the congestion window innovatively.

Hoe [99] proposed to enhance TCP slow start performance by setting a better initial value of *ssthresh* to be the estimated value of bandwidth delay product which is measured by the packet pair method. There are also some other works that focus on improving the estimate of *ssthresh* with a more accurate measurement of bandwidth delay product. Aron and Druschel [100] proposed to use multiple packet pairs to iteratively improve the estimate of *ssthresh*. *Paced Start* [101] uses packet trains to estimate the available bandwidth and *ssthresh*. These methods avoid TCP from prematurely switching to the congestion avoidance phase, but it may suffer temporary queue overflow and multiple packet losses when the bottleneck buffer is not big enough as compared to the bandwidth delay product. *Adaptive Start* [102] was proposed to reset *ssthresh* repeatedly to a more appropriate value by using eligible rate estimation. Adaptive start increases the congestion window (*cwnd*) size efficiently without packet overflows.

Several TCP slow start enhancements focus on intelligent adjustment of the congestion window size. At the beginning of the transmission, the exponential increase of the congestion window size is necessary to increase the bandwidth utilization quickly. However, it is too aggressive as the connection nears its equilibrium, leading to multiple packet losses. *Smooth Start* [103] improves the TCP slow start performance as the congestion window size approaches the connection equilibrium by splitting the slow-start into two phases, filling phase and probing phase. In the filling phase, the congestion window size is adjusted the same manner as traditional slow start, while it is increased more slowly in the probing phase. How to distinguish these phases is not addressed in smooth start. An additional threshold *max_ssthresh* is introduced in *Limited Slow Start* [104]. The congestion window size doubles per RTT, the same as traditional slow start, if the congestion window size is smaller than *max_ssthresh*; otherwise, the

congestion window size is increased by a fixed amount of *max_ssthresh* packets per RTT. Limited slow start reduces the number of drops in the TCP slow start phase, but *max_ssthresh* is required to be set statistically prior to starting a TCP connection.

Lu *et al.* [105] proposed a sender-side enhancement to slow-start by introducing a two-phase approach, linear increase and adjustive increase, to probe bandwidth more efficiently with TCP Vegas congestion-detection scheme. A certain threshold of queue length is used to signaling queue build-up. In the linear increase phase, TCP is started in the same manner as traditional slow start until the queue length exceeds the threshold. The congestion window size increment slows down to one packet per RTT to drain the temporary queue to avoid buffer overflow and multiple packet losses. Upon sensing the queue below the threshold, the sender enters the adjustive increase phase to probe for the available bandwidth more intelligently.

4) Congestion Control Enhancement: TCP variants have been developed for wide-area transport to achieve Gbps throughput levels. TCP Vegas [106] uses network buffer delay as an implicit congestion signal as opposed to drops. This approach may prove to be successful, but is challenging to implement. In wireless and wired-wireless hybrid networks, TCP-Jersey [107–109] enhances the available bandwidth estimations to improve the TCP performance by distinguishing the wireless packet losses and the congestion packet losses. By examining several significant TCP performance issues, such as unfair bandwidth allocation and throughput degradation, in wireless environments with window control theory, Hiroki *et al.* [110] found that the static additive-increase multiplicative-decrease (AIMD) window control policy employed by TCP is the basic cause for its performance degradation. In order to overcome the performance degradation caused by the static AIMD congestion window control, explicitly synchronized TCP (ESTCP) [110] was proposed by deploying a dynamic AIMD window control mechanism, which integrates the feedback information from networks nodes. High-speed TCP [111] modifies the congestion control mechanism with large congestion windows, and hence, High-speed TCP window will grow faster than standard TCP and also recover from losses more quickly. This behavior allows High-speed to quickly utilize the available bandwidth in networks with large bandwidth delay products. A simple sender side alteration to the TCP congestion window update algorithm, Scalable TCP [112], was proposed to improve throughput in highs-speed WANs. Numerical evaluation of the congestion control method of Scalable TCP and its impacts on other existing TCP versions were reported in [113].

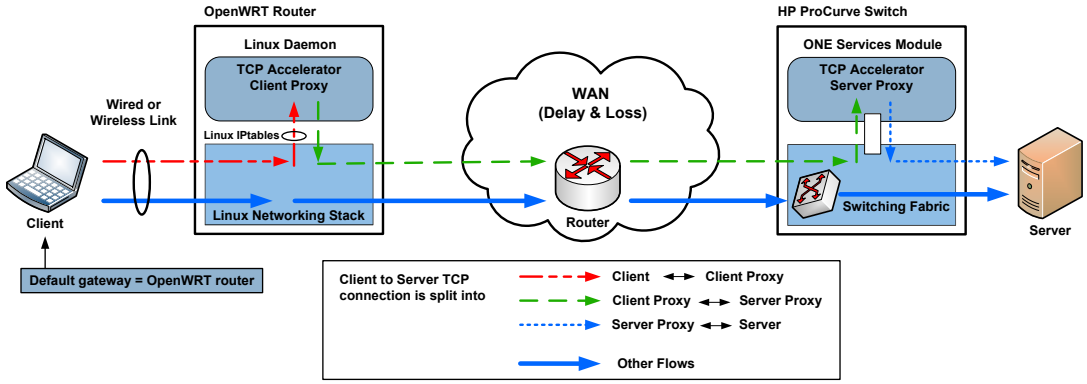
Fast TCP [114] is a TCP congestion avoidance algorithm especially targeted at long-distance and high latency links. TCP Westwood [115] implements a window congestion control algorithm based on a combination of adaptive bandwidth estimation strategy and an eligible rate estimation strategy to improve efficiency and friendliness tradeoffs. Competing flows with different RTTs may consume vastly unfair bandwidth shares in high-speed networks with large delays. Binary Increase Congestion Control (BIC) TCP [116] was proposed by taking RTT unfairness, together with TCP friendliness and bandwidth scalability, into consideration for TCP congestion control algorithm design. Another TCP friendly high speed TCP variant, CUBIC [117], is the current default TCP algorithm in Linux. CUBIC modifies the linear window growth function of existing TCP standards to be a cubic function in order to improve the scalability of TCP over fast and long distance networks. Fast TCP [114], BIC TCP [116], and CUBIC TCP [117] estimate available bandwidth on the sender side using intervals of Ack packets. However, the interval of Ack packets is influenced by traffic on the return path, thus making the estimation of the available bandwidth difficult and inaccurate.

5) TCP Transparent Proxy: The long delay is one of the main root causes for the TCP performance degradation over WANs. TCP transparent proxy involves breaking of long end-to-end control loops to several smaller feedback control loops by intercepting and relaying TCP connections within the network. The decrease in feedback delay accelerates the reaction of TCP flows to packet loss more quickly; hence, the accelerated TCP flows can achieve higher throughput performance.

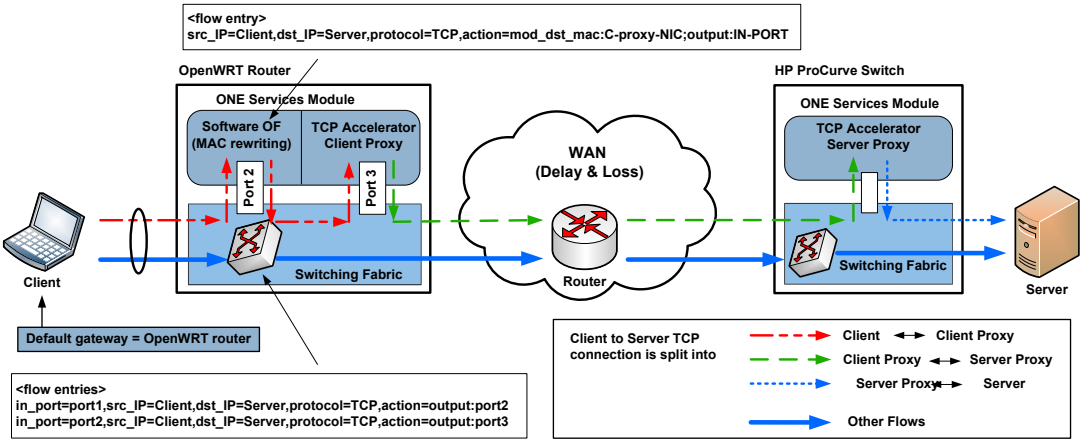
The snoop protocol [69] employs a proxy, normally a base station, to cache packets and perform local retransmissions across the wireless link to improve TCP performance by monitoring TCP acknowledgement packets. No changes are made to TCP protocols at the end hosts. Snoop TCP does not break the end-to-end TCP principle.

Split TCP connections have been used to cope with heterogeneous communication media to improve TCP throughput over wireless WANs (WWAN). Indirect-TCP (I-TCP) [70] splits the interconnection between mobile hosts and any hosts located in a wired network into two separate connections, one over the wireless medium and another over the wired network, at the network boundary with mobility support routers (MSRs) as intermediaries to isolate performance issues related to the wireless environment. Ananth *et al.* [71] introduced the idea of implementing a single logical end-to-end connection as a series of cascaded TCP connections. They analyzed the characteristics of the throughput of a split TCP connection analytically and proved the TCP throughput improvement by splitting the TCP connection. A flow aggregation based transparent TCP acceleration proxy was proposed and developed in [72] for GPRS network. The proxy splits TCP connections into wired and wireless parts transparently, and also aggregates the connections destined to the same mobile hosts due to their statistical dependence to maximize performance of the wireless link while inter-networking with unmodified TCP peers.

A Control for High-Throughput Adaptive Resilient Transport (CHART) system [118] was designed and developed by HP and its partners to improve TCP/IP performance and service quality guarantees with a careful re-engineering Internet layer 3 and layer 4 protocols. The CHART system enhances TCP/IP performance through two principal architectural innovations to the Internet Layer 3 and Layer 4 protocols. One is the fine-grained signaling and sensing within the network infrastructure to detect link failures and route around them. The other one is the explicit agreement between end hosts and the routing infrastructure on transmission rate, which will



(a) Client proxy built in the OpenWRT software router.



(b) Client proxy integrated in the hardware switch.

Fig. 9. Transparent TCP Acceleration proxy system [73].

permit the end hosts to transmit at the agreed rate independent of loss and delay. To achieve these two innovations, CHART couples a network-wide real-time network monitoring service, a new generation of network elements which monitor and control flows to explicitly assign available bandwidth to new and existing flows, and a new TCP driver, TCP-Trinity, which implements the explicit-rate protocol on the end hosts to accelerate data transmission by bypassing the slow-start and congestion avoidance phases of data transfer. Either manual modification on the end-hosts or route configuration changes are required by the original CHART TCP accelerators.

In order to realize transparent TCP acceleration, a network integrated transparent TCP accelerator proxy [73] was implemented on HP's ONE (Open Network Ecosystem) services blade that can be added to the switch chassis, and on a wireless OpenWRT platform with the CHART explicit-rate-signaling based TCP accelerator proxy. The network integrated TCP transparent acceleration is provided with flow redirection. Figure 9 illustrates the network integrated transparent TCP acceleration proxy system with client proxy built in the OpenWRT software router and a hardware layer 2 switch, respectively. As shown in these two figures, a single TCP connection is splitted into three connections by the CHART TCP acceleration system, namely, client to client proxy connection, client proxy to server proxy connection, and server proxy to server connection. The client proxy intercepts the first TCP SYN packet from the client and replies SYN-ACK to the client and initiates another TCP connection to the server proxy, which replies SYN-ACK to the client proxy and issues the final TCP connection to the server by receiving the TCP SYN packet from the client proxy. The client proxy and server proxy accelerate TCP connections between them via modifications to the TCP congestion control mechanism and explicit signaling of available bandwidth information. As depicted in Figure 9(a), the OpenWRT router is configured to use Linux IP tables to filter out the interested client-to-server TCP packets from the internal forwarding path, and redirect them to a TCP accelerator client proxy. While for the hardware switch integrated proxy shown in Figure 9(b), OpenFlow is used to run transparent TCP accelerator client proxy on the hardware switch. OpenFlow can classify packets based on 10 tuples, and thus TCP flows that require processing by the CHART proxy can be precisely selected. Owing to the limitations of OpenFlow on rewriting the destination MAC address, the selected packets are redirected to an OpenFlow software switch to rewrite the packet MAC address, and then re-inject into the network. The rewritten MAC address is the TCP accelerator client proxy input port MAC address; thus, the packets are then forwarded to the accelerator client proxy.

6) TCP Offload Engine: TCP Offload Engine (TOE) is a technology used in network interface cards (NIC) to offload TCP processing to the network controller. Moving complex

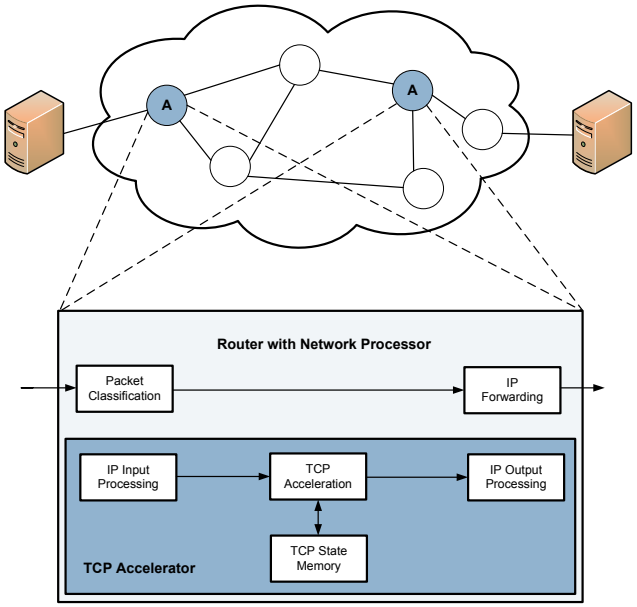


Fig. 10. System architecture of TCP acceleration nodes with network processors [74, 75].

TCP processing from end-systems into specified networking hardware reduces the system load and frees up resources. The experiment evaluation in [119] provided some insights into the effectiveness of the TOE stack in scientific as well as commercial domains.

A transparent TCP acceleration technique was proposed in [74, 75] by using network processors to increase TCP throughput inside the network without requiring any changes in end-system TCP implementations, and is thus undetectable to the end-system. Figure 10 shows the architecture of this TCP accelerator implemented on a network processor. The circles labeled by "A" are the acceleration nodes which split a long end-to-end TCP connection into several small TCP connections. Two packet forwarding paths are implemented in the acceleration nodes. Non-TCP packets or packets that cannot be accelerated due to resource constraint will be forwarded through "Path A" without any modification. Hence, packet classification is required to distribute packets to go through TCP acceleration or not. TCP accelerator involves IP input process, TCP acceleration, IP output processing, and a large TCP state memory storage. Network processors equipped with this transparent TCP accelerator can opportunistically act as TCP proxies by terminating TCP connections and opening a new connection towards the destination.

C. CIFS Optimization

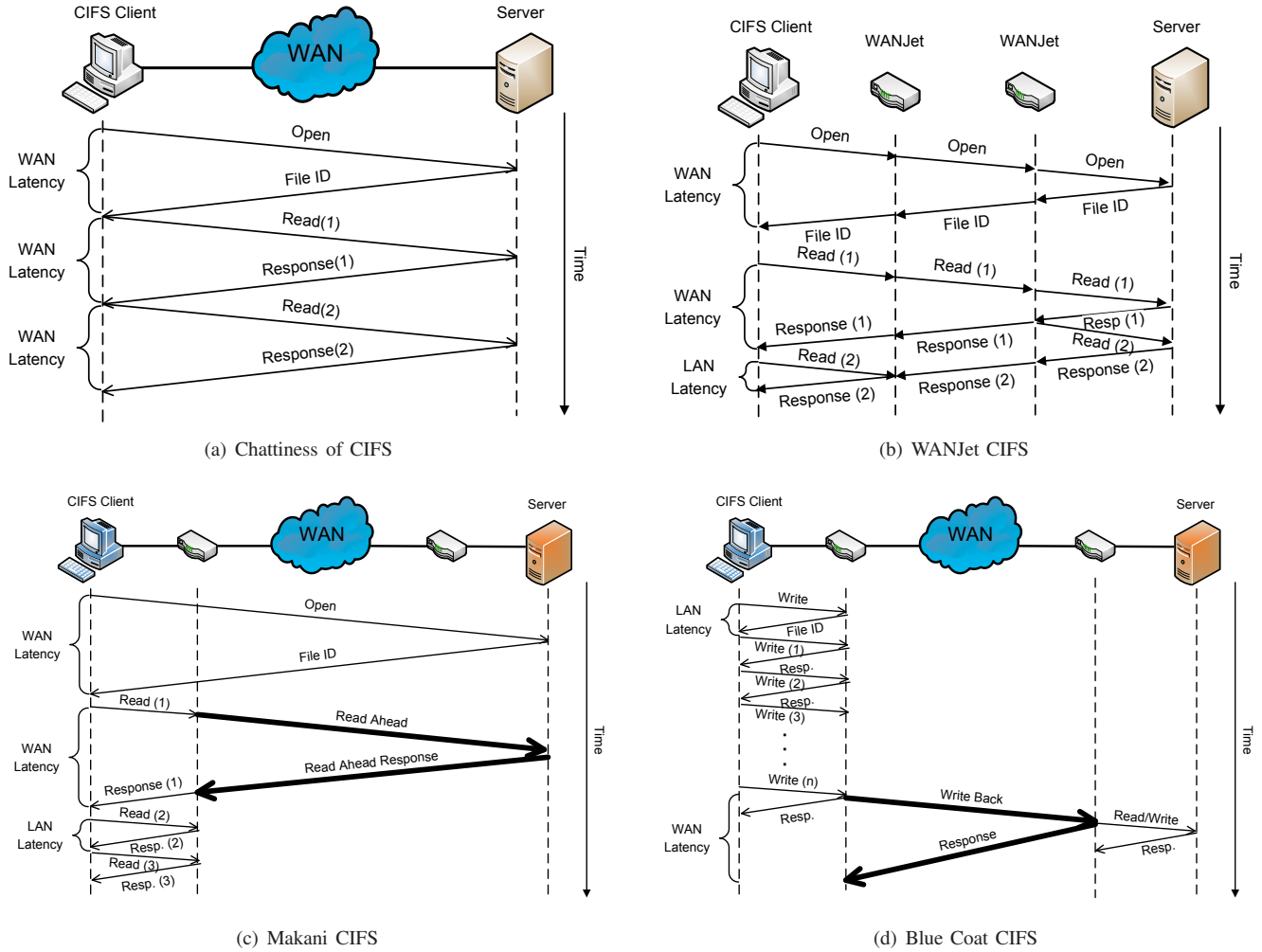
CIFS [82], also known as SMB [83], is a protocol developed by Microsoft for remote file access. CIFS defines the standard way that client systems share files across the Internet or other IP based networks. CIFS is a "chatty" protocol and not designed for high latency WAN environments. As has been pointed out, CIFS operates very poorly over a WAN [120]. The fundamental reason is that each CIFS request requires a response before the next request is sent to the CIFS server, implying that a large number of back and forth transactions are required to complete a request, and therefore, CIFS is redundant by generating multiple requests for a single file and the performance of CIFS decreases with the increase of WAN latency as shown in Figure 11(a). The small limitations of the windows client reading size, normally 4KB, will even worsen the CIFS performance since the smaller the reading size the more round-trips are required when a client is accessing a file from a server. Furthermore, CIFS was designed without considering bandwidth limitations, while the available bandwidth in WANs is normally limited. The redundant CIFS traffic over a bandwidth limited WAN deteriorates the CIFS performance further.

Many CIFS accelerators have been developed based on one or several of the following acceleration techniques:

- Caching: Respond to repeating client request without accessing CIFS servers.
- Predicting: Recognize well-known CIFS sequences, and act in advance, e.g., Read Ahead and Write Back, to reduce CIFS clients experienced latency from the WAN latency to the LAN latency.
- Read-ahead and Write-back: Read-ahead and Write-back can be implemented in reading (downloading) or writing (saving) a file from or to a server, respectively, to minimize the CIFS response time and improve CIFS performance consequently.
- Accumulating: Accumulate data in bigger messages to reduce the request messages.

The F5 WANJet CIFS acceleration [121] utilizes Transparent Data Reduction (TDR) [122], which is a two stage compression process to maximize bandwidth savings while minimizing processing latency, to reduce the latency experienced by the CIFS client from WAN latency to LAN latency as illustrated in Figure 11(b). In order to download a file from a CIFS server, the CIFS client issues an "open" CIFS request to the CIFS server, and the CIFS server responds with a file ID as the same sequence as in normal CIFS. Then, the CIFS client issues the first read request and the CIFS server responds with data. This first transaction incurs the WAN latency. After the initial transactions, the WANJet system can determine whether one CIFS client is attempting a file download. Thus, the server side WANJet system begins pre-fetching data by generating read requests locally to the server, and the pre-fetched data will be sent to the client side WANJet. For the subsequent CIFS client requests, instead of getting data from the server it now gets the replies locally from the client side WANJet at LAN latency. Using a combination of directory pre-fetching and caching, the WANJet system can greatly improve the response time of directory browsing.

Makani CIFS acceleration [123] implements a CIFS proxy that predicts future CIFS related transactions. If an appliance determines that a certain CIFS transaction is likely to occur, it pre-fetches data and temporarily stores it in the cache, which will be deleted once the pre-fetched data is referenced. After the initial transaction, the client side appliance can determine whether the CIFS client is attempting a file download; it starts pre-fetching data aggressively by generating read requests to the server. The server side appliance compresses the pre-fetched data and sends it to the client-side appliance. The



(a) Chattiness of CIFS

(b) WANJet CIFS

(c) Makani CIFS

(d) Blue Coat CIFS

Fig. 11. CIFS Acceleration Illustration

pre-fetched data is sent to the client side appliance and stored temporarily in anticipation of requests from the CIFS client; hence, CIFS appliances reduce the latency experienced by the CIFS client from WAN latency to LAN latency. Figure 11(c) illustrates that the Makani CIFS acceleration can reduce the latency experienced by the CIFS client from WAN latency to LAN latency when a client is accessing a file from the server. In the case that a CIFS client is writing a file to a CIFS server, the client side appliance responds locally to the CIFS client's write requests and passes the data to the server side appliance at WAN link speed to complete the write operation. Like WANJet, Makani CIFS accelerators can greatly improve the response time of directory browsing by using a combination of directory prefetching and caching.

Visuality System Ltd. developed a fully transparent embedded CIFS proxy solution, CAX (CIFS accelerator) [76], to improve user experience when accessing remote Windows file servers across the WAN by employing caching, predicting and accumulating. CAX can be embedded into network hardware including routers, hardware accelerators, and other products.

Blue Coat CIFS optimization [77] implements read-ahead and write back technique to reduce the latency associated with CIFS in reading or writing a file from or to a server, respectively. After having received a client CIFS read request, the Blue Coat appliance can recognize and anticipate future reads based on its knowledge of the CIFS protocol. Instead of making hundreds or thousands of small requests as the client is required to do, the Blue Coat appliance combines many smaller read requests into a single more efficient Read Ahead bulk request to save round-trips to the server by taking advantage of the ability of the CIFS protocol to support 60KB readings to retrieve the data in large chunks. It can further accelerate file retrieval by issuing multiple read requests to the server in parallel. The latency often occurs when writing or saving a file to a server since the client must wait until it receives acknowledgement from the server before sending additional data. Figure 11(d) depicts how the CIFS client writes a file to a server at LAN speed with Blue Coat CIFS protocol optimization. To minimize the response time when writing or saving a file to a server, the Blue Coat CIFS optimization implements Write Back. With Write Back, the client-side Blue Coat appliance acknowledges the client as if it was the server, allowing the client to write the file at LAN speed. Then, the client-side Blue Coat appliance will pass the

file to the server-side Blue Coat appliance, which is able to write the file to the server as it receives the data from the client, without requiring acknowledgement from the sever.

D. MAPI Optimization

MAPI [84] was originally designed by Microsoft to develop messaging applications. MAPI is also a "chatty" protocol and experiences low efficiency over a WAN as illustrated in Figure 12. A sender cannot send the next block of MAPI data before it receives the acknowledgement for the last block of data. Thus, the delay associated with client requests is bounded by the WAN latency; therefore, email application users often experience debilitating response time not only while sending and receiving e-mails, but also while accessing group message folders or changing calendar elements.

With MAPI protocol optimization, Makani appliances [78] retrieve MAPI data before clients request them by acting as a proxy between the client and server. Makani appliances implement split-proxy and read-ahead stream optimization to effectively reduce delays associated with the waiting time for data retrieval in the MAPI protocol. Makani appliances can terminate MAPI user requests as if they are severs, initiate connections to other Makani appliances across the WAN by implementing split-proxy, and respond to the client MAPI request as a remote server. Makani's MAPI optimization implements aggressive read-ahead stream optimization to minimize user response time. The client side Makani appliance monitors the first MAPI ReadStream request, and then reads the full stream ahead and stores it in the local file system. Thus, the latency is reduced, and consequently performance is improved.

Blue Coat's MAPI Optimization [79] minimizes the user response time by encapsulating multiple messages into large chunks to optimize the number of round-trips over the WAN. In order to further decrease the response time, Blue Coat's MAPI optimization employs the Keep-alive feature to stay connected and continue to retrieve data even after users have logged off. With Keep-alive, the local and remote appliances maintain "warm" connections between them to retrieve user e-mail and to populate the byte cache dictionaries of both proxies on each side of the WAN. Therefore, the clients can access the email application at LAN speed since the user email data has already been cached locally at the client side appliance.

E. Session Layer Acceleration

Several studies have presented two main session layer acceleration techniques: DNS- and URL- rewriting techniques [80], and parse and push techniques [81]. The performance of these session layer optimization techniques has been investigated in [124]. The main purpose of session layer optimization is to reduce the number of DNS lookups and the number of TCP connections at the client web browsers.

1) URL and DNS Rewriting: Webpages usually contain several embedded objects hosted under different domain names, and so the web browser has to perform several DNS queries for these domain names and set up at least one TCP connection to the URL server in each domain name for the

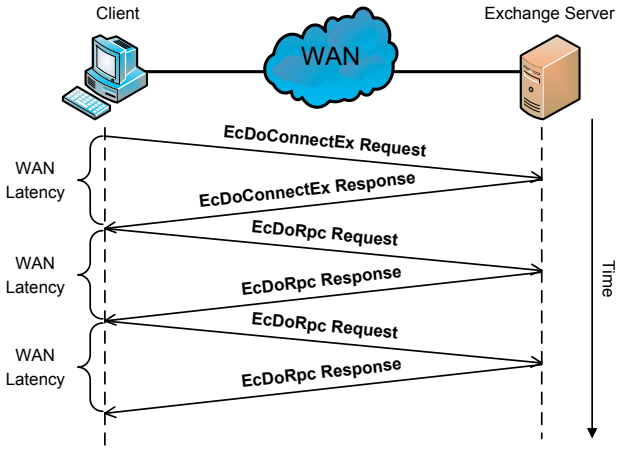
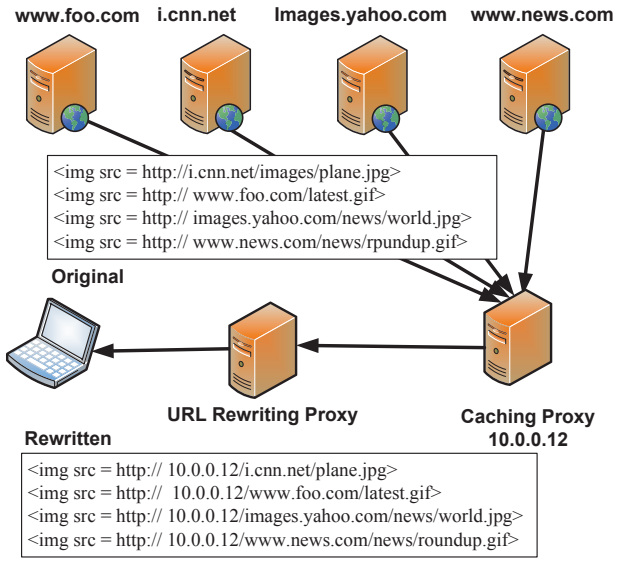


Fig. 12. Chattiness of MAPI

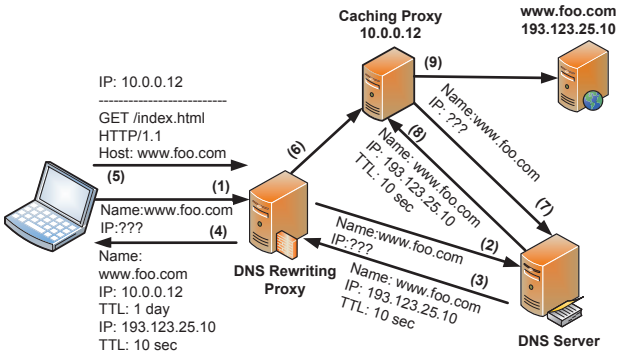
embedded objects. The large RTT in WANs increases the delay incurred by DNS lookups.

The main idea of URL-rewriting proposed in [80] is that the URLs for embedded objects in one web page are prefixed with the IP address of a caching proxy. Thus, no DNS requests are made by the client browser for the embedded URLs except the one that resolves the domain name of the server that hosts the top level page. As an example illustrated in Figure 13(a), URL rewriting is performed by a URL rewriting proxy closer to the client. The URL rewriting proxy intercepts the HTTP request and response for a top level web page transparently, and rewrites the URLs of the embedded objects by prefixing them with the IP address of a caching proxy, which could be located in the same or different machines. For example, the embedded URLs in the responding HTML file from the origin server www.foo.com is prefixed by the URL rewriting proxy with the IP address of the caching proxy, which is 10.0.0.12 in this example. Therefore, the following embedded object retrievals will be directed to the caching proxy server without any DNS lookups at the client browser and TCP connections to the origin servers to retrieve the embedded objects. The DNS requests for the embedded objects are made by the caching proxy if needed. Thus, the number of both DNS queries and TCP connections at the client browser are reduced.

Whenever a web browser makes a DNS request for a domain name, DNS rewriting proxy intercepts the DNS response and inserts the IP address of the caching proxy to the top of the original responded IP address list. Hence, the browser will attempt to connect to the caching proxy first. An example of DNS rewriting process to browse the web page http://www.foo.com/index.html is illustrated in Figure 13(b). In this example, DNS rewriting proxy inserts the IP address of the caching proxy 10.0.0.12 to the top of the DNS response. Then, the browser makes a HTTP GET request to the caching proxy to fetch /index.html. The caching proxy will make a DNS query to the DNS server directly to resolve the requested domain name and connects to the origin server to retrieve /index.html. Since all HTTP requests for top level web pages are directed to the same caching proxy, only one TCP connection is required to be set up between the client and the caching proxy, and it can be reused to retrieve multiple top level pages and their embedded objects.



(a) Example of URL rewriting



(b) Example of DNS rewriting

Fig. 13. Example of URL and DNS rewriting [80].

2) Parse and Push: The embedded objects are requested after the browser parses the HTML file. So, a round trip delay is normally incurred before transferring these embedded objects. The parse and push mechanism [81] enables the proxy server to pro-actively parse the HTML file and start to push the embedded objects towards the client. Thus, the web page download time can be reduced.

F. SSL/TLS Acceleration

Secure Socket Layer (SSL) and Transport Layer Security (TLS) are widely deployed cryptographic protocols for providing communication security over the Internet. Apostolopoulos *et al.* [125] showed that SSL/TLS handshake processing can be very processor intensive even without the cryptographic operations, and SSL/TLS can decrease the number of HTTP transactions handled by web servers up to two orders of magnitude. The extra latency introduced by SSL/TLS handshake messages, both from the serialized message exchange, as well as the cryptographic operations required for signing and encrypting those messages, was examined in [126]. The experimental results reported in [127] also distinctly show the inherent latency imposed by the SSL/TLS handshake.

1) Fast Cryptography Algorithms and Implementations: Several SSL handshake acceleration approaches involve techniques for speeding up encryption/decryption operations. Four fast RSA variants, namely, batch RSA, multi-prime RSA, multi-power RSA, and rebalanced RSA, have been examined to speed up RSA decryption in [128]. Batch RSA and two multi-factor RSA techniques, multi-prime RSA and multi-power RSA, are fully backward-compatible. Multiple public keys and certificates are required to be obtained and managed by the batch RSA algorithm. As compared to other three fast RSA variants, rebalanced RSA can achieve a large speedup, but only works with peer applications. By implementing batch RSA in an SSL web server, the performance of SSL's handshake protocol can achieve by up to a speedup factor of 2.5 for 1024-bit RSA keys as shown in [129]. There also exist numerous fast cryptography algorithms [130, 131].

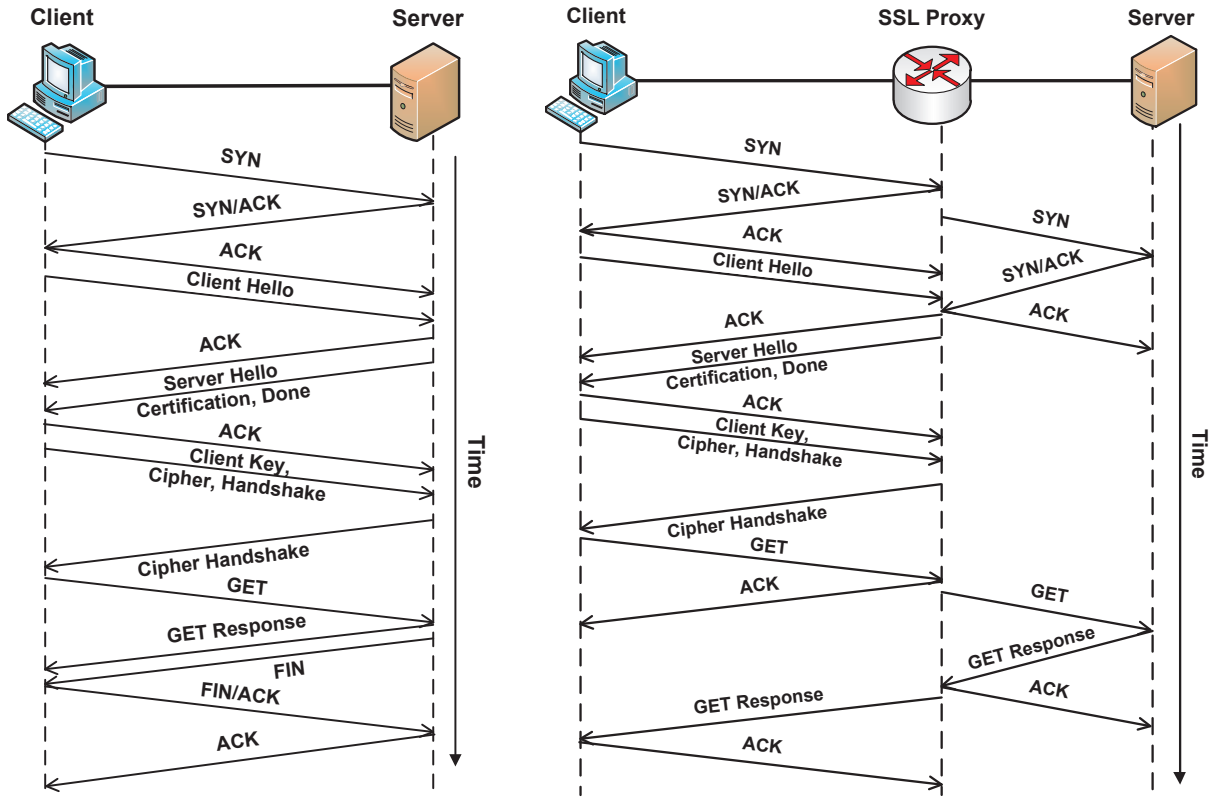
Elliptic Curve Cryptgraphy (ECC) can accelerate key computations with smaller key sizes while providing equivalent security. The performance impacts of ECC on SSL was studied in [132], which indicates that ECC implemented on an Apache web server enables 11%-31% more HTTPS requests per second handling over RSA. Fast ECC implementations with FPGA and hardware were reported in [133] and [134], respectively. To reduce complex certificate management overheads and long handshake latency in TLS, several identity-based cryptography primitives, such as identity-based encryption, identity-based signature, identity-based signcryption and identity-based authenticated key agreement can be adaptively applied to the TLS handshake protocol as illustrated in [135].

In order to alleviate server load, a cryptographic computation RSA-based re-balancing scheme between SSL clients and servers, called client-aided RSA (CA-RSA), was proposed in [136]. CA-RSA is achieved by adapting server-aided RSA (SA-RSA) to SSL/TLS settings with re-assigning SSL clients as "servers" while overloaded servers as "weak clients". The main idea of CA-RSA is to shift some computational load from the server to clients. Experimental results demonstrate that CA-RSA can speed up private-key computation by a factor of between 11 to 19 over plain RSA, depending on the RSA key size.

2) Compression to SSL/TLS: Introducing general purpose compression algorithms into SSL/TLS was reported in [137], and it was showed that the average text files transfer time can be improved especially for narrow bandwidth communication lines with compression to SSL/TLS. TLS allows clients and servers to negotiate selection of a lossless data compression method as part of the TLS Handshake Protocol. The compression methods associated with DEFLATE and Lempel-Ziv-Stac (LZS) lossless data compression algorithms for use with TLS was described in [138] and [139], respectively.

3) SSL/TLS Offload: It has been shown in several works that purely software-based cryptographic implementations are very costly, and therefore, a majority of SSL/TLS accelerators today focus on offloading the processor-intensive encryption algorithms involved in SSL/TLS transactions to a hardware accelerator. There are generally two TLS/SSL offload paradigms, server-resident and network-resident SSL/TLS hardware accelerators.

Server-resident TLS/SSL hardware accelerators are typically implemented in standard PCI cards, either as stand-alone devices, or integrated into network interface cards (NIC)



(a) Without SSL accelerator (b) With SSL accelerator proxy

Fig. 14. Examples of SSL handshake in a HTTP transaction.

or system chipsets. Numerous vendors offer SSL accelerator cards with cryptographic offload solutions, e.g., Broadcom, CAI Networks, Cavium Networks, Silicom, THALES, and IBM. Broadcom BCM5821 can achieve up to 4000 SSL/TLS transactions per second. Jang *et al.* [140] proposed an SSL-accelerator with Graphics Processing Units (GPUs) working as a proxy for web servers to offload the server-side cryptographic operations from CPUs. In order to achieve low latency and high throughput simultaneously, an opportunistic offloading algorithm was developed to balance the load between CPU and GPU in SSL processing depending on the load. The cryptographic operations will be handled by CPU for low latency when the load is light. As the number of pending cryptographic operations increases, cryptographic requests will pile up in the queue and cryptographic operations will be offloaded to GPUs to improve throughput with GPU's parallel executions. Preliminary results show that GPUs accelerate cryptographic operations with small latency overhead while significantly boosting the throughput at the same time. Berson *et al.* [141] demonstrated that public key cryptography can be provided as a network service over untrusted networks with a cryptoserver equipped with multiple cryptography accelerators, and thus the utilization of cryptographic accelerators can be improved by sharing them among many clients.

Network-resident SSL hardware accelerators normally comprise a multitude of switches, routers, load balancers, and custom network appliances. Many load balancer vendors, such as A10 Networks, Array Networks, Cisco Systems, F5 Networks, Citrix Netscaler, and Juniper Networks, offer solutions with SSL/TLS offload accelerators. Figure 14(a) shows the message flow for a typical HTTPS transaction with SSL/TLS handshake. From this figure, it is obvious that extra SSL/TLS handshake messages introduce latency, both from the serialized message exchange, as well as the cryptographic operations required for signing and encrypting those messages. Figure 14(b) illustrates the SSL/TLS acceleration with a network proxy within a HTTP transaction. The network resident SSL acceleration proxy terminates end-to-end TCP connection, SSL connection, and HTTP connection, and sets up two separate TCP connections and HTTP connections to clients and servers, respectively.

Ma and Bartoš [127] examined three SSL/TLS acceleration implementation schemes, namely, software-based functionality, server-resident hardware SSL accelerators, and centralized network-resident hardware SSL accelerators. They also discussed the trade-offs between the two different SSL/TLS offload techniques, SSL card accelerators and network proxies, and investigated their relationship with respect to the current performance of web services.

V. WAN Acceleration Products

Many companies provide WAN accelerators. The typical WAN acceleration products and the major WAN optimization techniques associated with them are listed in Table II.

A. ApplianSys

By caching frequently visited web pages locally, ApplianSys CACHEBOX web caching appliance dramatically im-

TABLE II
WAN ACCELERATION PRODUCTS OVERVIEW

Company	Typical Products	Major WAN Acceleration Techniques
ApplianSys	CacheBox	Bandwidth optimization, pre-caching
Aryaka Networks	Cloud-based application acceleration and WAN optimization	Compression, deduplication, protocol optimization (TCP, CIFS, MAPI, FTP), and guaranteed QoS
Blue Coat	ProxySG appliance, ProxyClient, Director, PacketShaper	Bandwidth management, protocol optimization, byte caching, object caching, compression
Cisco	Wide Area Application Services (WAAS) appliance, virtual WAAS, WAAS mobile, WAAS Express	Compression, deduplication, protocol optimization (HTTP, TCP, CIFS, MAPI), SSL optimization, NFS acceleration
Citrix	NetScaler, Branch Repeater	Layer 4 Load balancing, Layer 7 content switching, TCP optimization, HTTP compression, web optimization, SSL acceleration, and traffic management
Cyberoam	Unified Threat Management (UTM) platform	Bandwidth management, load balancing
Expand Networks	Appliance accelerator (Datacenter, regional office, small/satellite office), virtual accelerator (datacenter, branch office), mobile accelerator, Expandview management platform	TCP Acceleration, byte level caching, dynamic compression, Layer 7 QoS, application and protocol acceleration, wide area file services.
Ipanema Technologies	WAN Governance	Flow optimization, TCP optimization, Layer 7 CIFS optimization, QoS
Juniper Networks	WXC series application acceleration platform	Compression, caching, TCP acceleration, SSL optimization, application and protocol specific acceleration
Riverbed Technology	Steelhead appliance, Steel mobile software, virtual Steelhead appliance, WhiteWater appliance	Data deduplication, protocol optimization, and application layer protocol latency optimizations
Silver Peak Systems	NX appliance, VX appliance, VRX appliance	Deduplication, loss mitigation, QoS, and latency mitigation.
Exinda	UPM, ExOS 6.0	TCP optimization, application acceleration, universal caching, compression, intelligent acceleration, peer auto-discovery
UDcast	WANcompress, UDgateway	Caching and data compression
XipLink	XA appliance, XipOS	Data compression, TCP acceleration, HTTP acceleration, dynamic web cache

proves performance without the need to upgrade bandwidth, and offers solutions for a wide range of web caching requirements, such as reducing bandwidth costs and accelerating web access. CACHEBOX optimizes internet bandwidth usage by serving content stored in the LAN and accelerating performance on cached content.

B. Aryaka Networks

Aryaka offers a cloud-based solution for application acceleration and WAN optimization, which improves application performance as software-as-a-service (saas). The major employed acceleration techniques include compression, deduplication, application protocol optimization, TCP optimization, CIFS optimization, MAPI optimization, FTP optimization, and guaranteed QoS.

C. Blue Coat

The Blue Coat ProxySG appliances can be deployed at the core and edges of an enterprise's application delivery infrastructure to improve user productivity and reduce bandwidth expense. The main acceleration techniques employed by ProxySG include compression, bandwidth management, protocol optimization, and caching. ProxyClient enables remote acceleration to provide a LAN-like experience to remote users connecting over VPNs or WANs. Blue Coat Director delivers comprehensive, centralized WAN optimization management for a large network of ProxySG appliances. PacketShaper, together with the ProxySG appliances, can monitor and accelerate application performance for real-time protocols.

D. Cisco

Cisco WAAS software optimizes the performance of any TCP-based application across a WAN by a combination of WAN optimization, acceleration of TCP-based applications, and Cisco's Wide Area File Services (WAFS). WAAS implements comprehensive application optimization, application-specific protocol acceleration, bandwidth reduction through compression and deduplication, and TCP flow optimization. WAAS software can be delivered with WAAS appliances, Cisco services ready engine modules on Cisco integrated services routers, and some dedicated blades. Cisco virtual WAAS (vWAAS) is a virtual appliance that accelerates applications delivered from private and virtual private cloud infrastructures over WANs, through policy-based on-demand orchestration. Cisco WAAS mobile extends application acceleration benefits to mobile workers. WAAS mobile optimizes cellular, satellite, WiFi, WiMax, and DSL networks to reduce

timing variations, high latency, and noisy connections, and increases link resiliency. The application protocol optimizers employed by WAAS mobile reduce application round trips for file transfers, Outlook, enterprise web applications, and web browsing. Cisco WAAS Express leverages a combination of data redundancy elimination, data compression, and TCP flow optimization to maximize bandwidth optimization gains.

E. Citrix

Citrix NetScaler, a web application delivery appliance, optimizes application performance through advanced layer 4 load balancing, layer 7 content switching, TCP optimization, HTTP compression, web optimization, SSL acceleration, and traffic management. Citrix Branch Repeater, available as both a physical and a virtual appliance, optimizes the delivery of applications to branch and mobile users. Branch Repeater appliances are auto-discovered, auto-configured, and auto-tuned. Branch Repeater incorporates two Citrix HDX technologies, HDX IntelliCache, and HDX Broadcast, to optimize both hosted and streamed applications delivered to branches. HDX IntelliCache optimizes performance for multiple users accessing virtual desktops and applications from branch offices by locally caching and de-duplicating bandwidth intensive data, and by locally staging streamed application packages. HDX Broadcast provides a set of technologies that adaptively tune to real-time conditions to optimize network traffic.

F. Cyberoam

Cyberoam Unified Threat Management (UTM) appliances support 3G and WiMAX WAN connectivity which offers assured security over wireless WAN links. Cyberoam Bandwidth Management offers identity-based bandwidth control, preventing congestion and bandwidth abuse, and optimizing bandwidth and multiple link management that provides WAN redundancy and delivers assured WAN availability and reliable connectivity.

G. Expand Networks

WAN acceleration products provided by Expand Networks include appliance accelerator, virtual accelerator, and mobile client accelerator along with ExpandView management platform. They improve WAN performance by providing virtual bandwidth. Expand appliance accelerators are available in many sizes ranging from data center appliances, large to medium regional offices, branch offices, to satellite offices. By leveraging the existing virtual IT infrastructure in the datacenter and branch offices, WAN optimization can be deployed as a virtual appliance for datacenters and branch offices. Mobile accelerator transforms the economics of WAN optimization for smaller branch offices and mobile workers within medium to large sized businesses. ExpandView, a central management, reporting and alerting system, provides application fluent network statistics, optimization performance, bandwidth utilization and configuration management for appliance, and virtual and mobile accelerators. One comprehensive, multi-service platform with extensive management capabilities is developed by ExpandView to incorporate many WAN optimization techniques, including TCP acceleration, byte level caching, dynamic compression, layer 7 QoS, application and protocol acceleration, and wide area file services.

H. Ipanema Technologies

Ipanema WAN Governance extends application performance management and permits a continuous service level management through clear key performance indicators by implementing a combination of application visibility, QoS and control, WAN optimization, and dynamic WAN selection. Ipanema's WAN optimization utilizes both RAM and disk redundancy elimination to optimize all TCP and UDP applications flows. It also employs CIFS optimization to improve Microsoft file sharing efficiency over WAN. All Ipanema WAN optimization mechanisms are under the control of Ipanema's unique QoS and control innovation, and are automatically tuned by Ipanema's autonomic networking architecture.

I. Juniper Networks

The Juniper WXC series application acceleration platform provides a scalable way to speed up the delivery of client-server and web-based business applications and services over the WAN by recognizing and eliminating redundant transmissions, accelerating TCP and application-specific protocols, as well as prioritizing and allocating access bandwidth. The major acceleration techniques employed by the WXC series application acceleration platform include compression, caching, TCP acceleration, SSL optimization, and application acceleration.

J. Riverbed Technology

Riverbed Steelhead appliances accelerate a broad range of applications, including file sharing, exchange (MAPI), Lotus Notes, web, database, and disaster recovery applications. The Steelhead appliance employs data reduction, TCP acceleration, and application layer protocol latency optimizations to enhance application performance. Steelhead Mobile client software enables application acceleration to mobile users. With Riverbed Virtual Steelhead appliances, WAN optimization can be extended to deployments where physical hardware may not be conveniently suited. The WhiteWater appliance is designed for optimization towards cloud storage providers.

K. Silver Peak Systems

Typical Silver Peak WAN acceleration appliances include NX appliance, VX appliance, and VRX appliance. The VX appliances are the software versions of the NX appliances. The VRX appliances are the first and the only virtual WAN optimization devices designed for data center deployment. All of these appliances leverage a unique blend of WAN acceleration techniques, including compression, data deduplication, loss mitigation, QoS provisioning, and latency mitigation. Silver Peak WAN acceleration appliances employ network acceleration, especially on TCP and other protocol acceleration techniques, to minimize the effects of latency on application performance and significantly improve application response time across the WAN. These appliances also utilize a variety of Quality of Service (QoS) and traffic shaping techniques

to optimize traffic handling, including advanced queuing, scheduling, and standards-based packet-marking. One important feature of these appliances is that network memory is used for data deduplication. Network memory operates at the network layer and supports all IP-based protocols.

L. Exinda

Exinda Unified Performance Management (UPM) is a network appliance that integrates real-time network monitoring, reporting, traffic control, and application acceleration to effectively manage a WAN. The optimization techniques leveraged by UPM include TCP optimization, application acceleration, universal caching, compression, intelligent acceleration, and peer auto-discovery. Newly released ExOS 6.0 offers some new features, such as TCP efficiency monitoring and reporting, TCP health monitoring, service level agreement monitoring, and SSL/HTTPs acceleration.

M. UDCast

UDCast WANcompress offers bandwidth optimization to dramatically improve IP communications over satellite, wireless, wired and WiMAX networks across WAN, and reduces the high operating expenditure associated with satellite communications. Based on a combination of compression and caching techniques to eliminate redundant data, WANcompress can save on average 50% of the bandwidth on a given communications link. In order to perform real-time compression, WANcompress performs data matching at the byte level. The UDgateway service platform with WANcompress improves satellite communication performance.

N. XipLink

The XipLink XA series appliances work together with XA optimization software (XipOS) to deliver satellite and wireless optimization. XipOS maximizes bandwidth across wireless communication links by using extensions of the Space Communication Protocol Specification (SCPS). It offers 2-10 times bandwidth gain through streaming compression. It also utilizes TCP acceleration and Internet optimization through HTTP acceleration and dynamic web caching.

VI. Conclusion

In this paper, we have provided a detailed discussion on performance enhancement techniques over a WAN, especially with the focus on WAN optimization, also known as WAN acceleration. We first reviewed the obstacles to content delivery over a WAN. To overcome these challenges in the WANs, many WAN acceleration techniques have been proposed and developed. We have summarized most commonly used WAN optimization techniques, in particular, compression, data deduplication, caching, prefetching, and protocol optimization. Finally, we have presented commonly available WAN acceleration products and the major WAN optimization techniques associated with these acceleration products. We hope that this paper will serve as a valuable vehicle for further research on WAN optimization.

References

[1] P. Sevcik and R. Wetzel, "Improving Effective WAN Throughput for Large Data Flows," http://www.silver-peak.com/assets/download/pdf/Netforecast_wp_EffectiveThroughput.pdf, November 2008.

[2] T. Grevers Jr. and J. Christner, *Application Acceleration and WAN Optimization Fundamentals*. Indianapolis, IN: Cisco Press, 2007.

[3] S. Corner, http://www.submitcorner.com/Guide/Bandwidth/001.shtml.

[4] Akamai, http://www.akamai.com/dl/reports/Site_Abandonment_Final_Report.pdf, November 2006.

[5] ——, http://www.akamai.com/dl/whitepapers/ecommerce_website_perf_wp.pdf?curl=/dl/whitepapers/ecommerce_website_perf_wp.pdf&solcheck=1&.

[6] M. Al-laham and I. M. M. E. Emary, "Comparative study between various algorithms of data compression techniques," *International Journal of Computer Science and Network Security*, vol. 7, no. 4, April 2007.

[7] S. Sakr, "Xml compression techniques: A survey and comparison," *Journal of Computer and System Sciences*, vol. 75, no. 5, pp. 303 – 322, 2009.

[8] J. Shukla, M. Alwani, and A. Tiwari, "A survey on lossless image compression methods," in *Proc. 2nd International Conference on Computer Engineering and Technology*, April 2010, pp. 136–141.

[9] R. J. Clarke, "Image and video compression: a survey," *International Journal of Imaging Systems and Technology*, vol. 10, pp. 20–32, 1999.

[10] Z. Liu, Y. Saifullah, M. Greis, and S. Sreemanthula, "HTTP Compression Techniques," in *Proc. IEEE Wireless Communications and Networking Conference*, March 2005, pp. 2495 – 2500.

[11] N. Mandagere, P. Zhou, M. A. Smith, and S. Uttamchandani, "Demystifying Data Deduplication," in *Proc. Middleware Conference Companion*, Leuven, Belgium, 2008, pp. 12–17.

[12] Q. He, Z. Li, and X. Zhang, "Data Deduplication Techniques," in *Proc. International Conference on Future Information Technology and Management Engineering (FITME)*, Oct. 2010, pp. 430 –433.

[13] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur, "Single Instance Storage in Windows 2000," in *Proc. 4th conference on USENIX Windows Systems Symposium*, Seattle, Washington, Aug. 2000, pp. 13–24.

[14] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming Space from Duplicate Files in a Serverless Distributed File System," in *Proc. 22 nd International Conference on Distributed Computing Systems*, Vienna, Austria, July 2002, pp. 617–624.

[15] N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," in *Proc. SIGCOMM*, Stockholm, Sweden, 2000, pp. 87–95.

[16] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet caches on routers: the implications of universal redundant traffic elimination," in *Proc. SIGCOMM*, Seattle, WA, USA, 2008, pp. 219–230.

[17] A. Anand, V. Sekar, and A. Akella, "SmartRE: an architecture for coordinated network-wide redundancy elimination," *SIGCOMM Computer Communication Review*, vol. 39, pp. 87–98, August 2009.

[18] B. Aggarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese, "EndRE: an end-system redundancy elimination service for enterprises," in *Proc. 7th USENIX conference on Networked systems design and implementation*, San Jose, California, 2010, pp. 28–28.

[19] G. Lu, Y. Jin, and D. Du, "Frequency based chunking for data deduplication," in *Proc. Modeling, Analysis Simulation of Computer and Telecommunication Systems*, Aug. 2010, pp. 287 –296.

[20] S. Saha, A. Lukyanenko, and A. Yla-Jaaski, "Combiheader: Minimizing the number of shim headers in redundancy elimination systems," in *Proc. INFOCOM workshops on Computer Communications*, April 2011, pp. 798 –803.

[21] S. Quinlan and S. Dorward, "Venti: A New Approach to Archival Data Storage," in *Proc. 1st USENIX Conference on File and Storage Technologies (FAST)*, 2002, pp. 89 – 101.

[22] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li, "Decentralized Deduplication in SAN Cluster File Systems," in *Proc. USENIX Annual technical conference*, San Diego, California, June 2009.

[23] C. Constantinescu, J. Pieper, and T. Li, "Block Size Optimization in Deduplication Systems," in *Proc. Data Compression Conference*, 2009.

[24] J. J. Hunt, K.-P. Vo, and W. F. Tichy, "An Empirical Study of Delta Algorithms," in *Proc. SCM-6 Workshop on System Configuration Management*, 1996, pp. 49–66.

[25] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: a decentralized peer-to-peer web cache," in *Proc. 21th annual symposium on Principles of distributed computing*, Monterey, California, 2002, pp. 213–222.

[26] L. Xiao, X. Zhang, and Z. Xu, "On reliable and scalable peer-to-peer web document sharing," in *Proc. International Parallel and Distributed Processing Symposium (IPDPS)*, Fort Lauderdale, FL, April 2002, pp. 23–30.

[27] Z. Xu, Y. Hu, and L. Bhuyan, "Exploiting client cache: a scalable and efficient approach to build large web cache," in *Proc. 18th International Parallel and Distributed Processing Symposium (IPDPS)*, April 2004, pp. 55–64.

[28] A. Vakali, "Proxy Cache Replacement Algorithms: A History-Based Approach," *World Wide Web*, vol. 4, pp. 277–297, 2001.

[29] T. M. Kroeger, D. D. E. Long, and J. C. Mogul, "Exploring the Bounds of Web Latency Reduction from Caching and Prefetching," in *Proc. USENIX Symp. on Internet Technologies and Systems*, Dec. 1997.

[30] P. Rodriguez, C. Spanner, and E. Biersack, "Analysis of web caching architectures: hierarchical and distributed caching," *IEEE/ACM Trans. Netw.*, vol. 9, no. 4, pp. 404 –418, Aug. 2001.

[31] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical internet object cache," in *Proc. 1996 annual conference on USENIX Annual Technical Conference*, San Diego, CA, 1996.

[32] D.Wessels and K. Claffy, "Application of Internet cache protocol (ICP) version 2," http://tools.ietf.org/html/rfc2187, May 1997.

[33] A. Rousskov and D. Wessels, "Cache digests," in *Proc. 3rd Internaltional WWW Caching Workshop*, June 1998, pp. 272–273.

[34] R. Tewari, M. Dahlin, H. Vin, and J. Kay, "Design considerations for distributed caching on the internet," in *Proc. 19th International Conference on Distributed Computing Systems*, 1999, pp. 273 –284.

[35] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi, "Web caching with consistent hashing," in *Proc. eighth international conference on World Wide Web*, Toronto, Canada, 1999, pp. 1203–1213.

[36] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, pp. 281–293, June 2000.

[37] K. Leung, E. Wong, and K. Yeung, "Design of distributed video cache system on the internet," in *Proc. 23rd International Conference on Distributed Computing Systems Workshops*, May 2003, pp. 948 – 953.

[38] M. Rabinovich, J. Chase, and S. Gadde, "Not all hits are created equal: cooperative proxy caching over a wide-area network," *Computer Networks and ISDN Systems*, vol. 30, pp. 2253–2259, November 1998.

[39] J. Ravi, Z. Yu, and W. Shi, "A survey on dynamic web content generation and delivery techniques," *Journal of Network Computer Applications*, vol. 32, pp. 943–960, September 2009.

[40] M. Abrams, C. R. Standridge, G. Abdulla, E. A. Fox, and S. Williams, "Removal Policies in Network Caches for World-Wide Web Documents," in *Proc. SIGCOMM*, California, USA, 1996, pp. 293–305.

[41] J. Domenech, J. Sahuquillo, J. Gil, and A. Pont, "The Impact of the Web Prefetching Architecture on the Limits of Reducing User's Perceived Latency," in *Proc. IEEE/WIC/ACM International Conference on Web Intelligence*, Dec. 2006, pp. 740 –744.

[42] A. Balamash, M. Krunz, and P. Nain, "Performance analysis of a client-side caching/prefetching system for web traffic," *Comput. Netw.*, vol. 51, pp. 3673–3692, September 2007.

[43] V. N. Padmanabhan and J. C. Mogul, "Using predictive prefetching to improve world wide web latency," *SIGCOMM Computer Communication Review*, vol. 26, pp. 22–36, July 1996.

[44] T. Palpanas, "Web Prefetching Using Partial Match Prediction," Department of Computer Science, University of Toronto, Master's Thesis, March 1998, (available as Technical Report CSRG-376 http://www.cs.toronto.edu/themis/publications/webprefetch.pdf).

[45] J. Domènech, J. Gil, J. Sahuquillo, and A. Pont, "DDG: An Efficient Prefetching Algorithm for Current Web Generation," in *Proc. 1st IEEE Workshop on Hot Topics in Web Systems and Tech*, 2006.

[46] R. Kokku, P. Yalagandula, A. Venkataramani, and M. Dahlin, "NPS: A Non-interfering Deployable Web Prefetching System," in *Proc. 4th USENIX Symposium on Internet Technologies and Systems*, 2003.

[47] A. Balamash, M. Krunz, and P. Nain, "Performance analysis of a client-side caching/prefetching system for web traffic," *Comput. Netw.*, vol. 51, pp. 3673–3692, September 2007.

[48] X. Chen and X. Zhang, "Coordinated data prefetching by utilizing reference information at both proxy and web servers," *SIGMETRICS Performance Evaluation Review*, vol. 29, pp. 32–38, September 2001.

[49] C. Bouras, A. Konidaris, and D. Kostoulas, "Predictive Prefetching on the Web and Its Potential Impact in the Wide Area," *World Wide Web*, vol. 7, pp. 143–179, 2004.

[50] L. Fan, P. Cao, W. Lin, and Q. Jacobson, "Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance," in *Proc. ACM SIGMETRICS*, Atlanta, Georgia, 1999, pp. 178–187.

[51] J. Pitkow and P. Pirolli, "Mining longest repeating subsequences to predict world wide web surfing," in *Proc. 2nd conference on USENIX Symposium on Internet Technologies and Systems*, Boulder, Colorado, Oct. 11-14 1999.

[52] X. Chen and X. Zhang, "Popularity-based ppm: an effective web prefetching technique for high accuracy and low storage," in *Proc. International Conference on Parallel Processing*, 2002, pp. 296 – 304.

[53] R. R. Sarukkai, "Link prediction and path analysis using markov chains," in *Proc. 9th international World Wide Web conference on Computer networks*, Amsterdam, The Netherlands, 2000, pp. 377–386.

[54] M. Deshpande and G. Karypis, "Selective markov models for predicting web page accesses," *ACM Transactions Internet Technology*, vol. 4, pp. 163–184, May 2004.

[55] E. Markatos and C. Chronaki, "A top-10 approach to prefetching on the web," in *Proc. INET*, Geneva, Switzerland, 1998.

[56] A. Nanopoulos, D. Katsaros, and Y. Manolopoulos, "A data mining algorithm for generalized web prefetching," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 5, pp. 1155 – 1169, Sep.-Oct. 2003.

[57] A. Songwattana, "Mining web logs for prediction in prefetching and caching," in *Proc. Third International Conference on Convergence and Hybrid Information Technology*, vol. 2, 2008, pp. 1006 –1011.

[58] J. Dean and M. R. Henzinger, "Finding related pages in the world wide web," in *Proc. 8th international conference on World Wide Web*, Toronto, Canada, 1999, pp. 1467–1479.

[59] D. Duchamp, "Prefetching hyperlinks," in *Proc. USENIX Symposium on Internet Technologies and Systems*, Boulder, Colorado, 1999.

[60] T. Ibrahim and C.-Z. Xu, "Neural nets based predictive prefetching to tolerate www latency," in *Proc. 20th International Conference on Distributed Computing Systems (ICDCS)*, April 2000, pp. 636 –643.

[61] B. D. Davison, "Predicting web actions from html content," in *Proc. the thirteenth ACM conference on Hypertext and hypermedia*, College Park, Maryland, USA, 2002, pp. 159–168.

[62] A. Georgakis and H. Li, "User behavior modeling and content based speculative web page prefetching," *Data Knowl. Eng.*, vol. 59, pp. 770–788, December 2006.

[63] V. N. Padmanabhan and J. C. Mogul, "Improving http latency," *Computer Networks and ISDN Systems*, vol. 28, pp. 25–35, Dec. 1995.

[64] A. Abhari and A. Serbinski, "HTTP modification to reduce client latency," in *Proc. Canadian Conference on Electrical and Computer Engineering (CCECE)*, May 2008, pp. 1491–1496.

[65] J. Mickens, "Silo: exploiting javascript and dom storage for faster page loads," in *Proceedings of the 2010 USENIX conference on Web application development*, ser. WebApps'10, Boston, MA, 2010.

[66] I. Cidon, R. Rom, A. Gupta, and C. Schuba, "Hybrid tcp-udp transport for web traffic," in *Proc. IEEE International Performance, Computing and Communications Conference*, Feb. 1999, pp. 177–184.

[67] M. Rabinovich and H. Wang, "Dhttp: an efficient and cache-friendly transfer protocol for the web," *IEEE/ACM Trans. Netw.*, vol. 12, pp. 1007–1020, December 2004.

[68] F. Yang, Y. Dou, Z. Lei, H. Zou, and K. Zhang, "The Optimization of HTTP Packets Reassembly based on Multi-core Platform," in *Proc. 2nd IEEE International Conference on Network Infrastructure and Digital Content*, Sept. 2010, pp. 530 –535.

[69] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP Performance over Wireless Networks," in *Proc. 1st annual international conference on Mobile computing and networking*, Berkeley, California, 1995, pp. 2–11.

[70] A. Bakre and B. Badrinath, "I-TCP: indirect TCP for Mobile Hosts," in *Proc. 15th International Conference on Distributed Computing Systems*, 1995, pp. 136 –143.

[71] A.I. Sundararaj and D. Duchamp, "Analytical Characterization of the Throughput of a Split TCP Connection," Department of Computer Science, Stevens Institute of Technology, Tech. Rep., 2003.

[72] R. Chakravorty, S. Katti, C. J, and P. I, "Flow Aggregation for Enhanced TCP over Wide-Area Wireless," in *Proc. INFOCOM*, April 1-3 2003, pp. 1754 – 1764.

[73] J. Lee, P. Sharma, J. Tourrilhes, R. McGeer, J. Brassil, and A. Bavier, "Network Integrated Transparent TCP Accelerator," in *Proc. 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, April 2010, pp. 285 –292.

[74] T. Wolf, S. You, and R. Ramaswamy, "Transparent TCP Acceleration Through Network Processing," in *Proc. IEEE Global Telecommunications Conference (GLOBECOM)*, Dec. 2005, pp. 750–754.

[75] S. Ladiwala, R. Ramaswamy, and T. Wolf, "Transparent TCP Acceleration," *Computer Communication*, vol. 32, pp. 691–702, March 2009.

[76] M. Rabinovich and I. Gokhman, "CIFS Acceleration Techniques," in *Proc. Storage Developer Conference*, 2009.

[77] Blue Coat, "Blue Coat CIFS Protocol Optimization," http://www.bluecoat.com/doc/398%20-.

[78] Makani Networks, "Makani MAPI Protocol Optimization," http://www.makaninetworks.com/register/docs/makani-mapi.pdf.

[79] Blue Coat, "Blue Coat MAPI Protocol Optimization," http://www.bluecoat.com/doc/397%20-.

[80] P. Rodriguez, S. Mukherjee, and S. Ramgarajan, "Session level techniques for improving web browsing performance on wireless links," in *Proc. 13th international conference on World Wide Web*, New York, NY, USA, 2004, pp. 121–130.

[81] R. Chakravorty, A. Clark, and I. Pratt, "Gprsweb: optimizing the web for gprs links," in *Proc. 1st international conference on Mobile systems, applications and services (MobiSys)*, San Francisco, California, 2003, pp. 317–330.

[82] Common Internet File System (CIFS), http://msdn.microsoft.com/en-us/library/aa302188.aspx.

[83] Server Message Block, http://en.wikipedia.org/wiki/Server_Message_Block#NetBIOS.

[84] Microsoft, "Messaging Application Programming Interface (MAPI)," http://msdn.microsoft.com/en-us/library/aa142548(v=exchg.65).aspx.

[85] M. Machowinski, "WAN optimization market passes $1 billion in 2008, up 29%; enterprise router market down."

[86] J. Skorupa, "Forecast: Application Acceleration Equipment, Worldwide, 2007-2015, 1Q11 Updated," http://www.gartner.com/DisplayDocument?id=1577015, March 2011.

[87] Google, http://www.chromium.org/spdy/spdy-whitepaper.

[88] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol – HTTP/1.0," *IETF RFC 1945*, May 1996.

[89] R. Fielding *et. al*, "Hypertext Transfer Protocol – HTTP/1.1," *IETF RFC 2616*, June 1999.

[90] A. B. King, *Speed up Your Site: Web Site Optimization*. New Riders, 2003.

[91] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: findings and implications," in *Proc. 11th International Joint Conference on Measurement and Modeling of Computer Systems*, Seattle, WA, USA, 2009, pp. 37–48.

[92] J. Cleary and I. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Trans. Commun.*, vol. 32, no. 4, pp. 396 – 402, Apr. 1984.

[93] J. Domènech, J. A. Gil, J. Sahuquillo, and A. Pont, "Web prefetching performance metrics: a survey," *Perform. Eval.*, vol. 63, pp. 988–1004, October 2006.

[94] AKAMAI TECHNOLOGIES, "Akamai reveals 2 seconds as the new threshold of acceptability for ecommerce web page response times," http://www.akamai.com/html/about/press/releases/2009/press_091408.html, Sep. 14, 2009.

[95] V. Paxson, "Growth Trends in Wide-Area TCP Connections," *IEEE Network*, vol. 8, no. 4, pp. 8 –17, Jul/Aug 1994.

[96] E. Besson, "Performance of TCP in a Wide-Area Network: Influence of Successive Bottlenecks and Exogenous Traffic," in *Proc. Global Telecommunications Conference*, vol. 3, 2000, pp. 1798 –1804.

[97] M. Mirza, J. Sommers, P. Barford, and X. Zhu, "A Machine Learning Approach to TCP Throughput Prediction," *IEEE/ACM Trans. Netw.*, vol. 18, no. 4, pp. 1026 –1039, Aug. 2010.

[98] N. Rao, S. Poole, S. Hicks, C. Kemper, S. Hodson, G. Hinkel, and J. Lothian, "Experimental Study of Wide-Area 10 Gbps IP Transport Technologies," in *Proc. Military Communications Conference (MILCOM)*, Oct. 2009, pp. 1 –7.

[99] J. C. Hoe, "Improving the start-up behavior of a congestion control scheme for tcp," in *Proc. SIGCOMM*, Palo Alto, California, United States, 1996, pp. 270–280.

[100] M. Aron and P. Druschel, "Tcp: Improving start-up dynamics by adaptive timers and congestion control," Rice University, Tech. Rep. TR98-318, 1998.

[101] N. Hu and P. Steenkiste, "Improving tcp startup performance using active measurements: algorithm and evaluation," in *Proc. 11th IEEE International Conference on Network Protocols*, Nov. 2003, pp. 107 – 118.

[102] R. Wang, G. Pau, K. Yamada, M. Sanadidi, and M. Gerla, "Tcp startup performance in large bandwidth networks," in *Proc. INFOCOM*, vol. 2, March 2004, pp. 796 – 805.

[103] H. Wang and C. Williamson, "A new scheme for tcp congestion control: smooth-start and dynamic recovery," in *Proc. Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, July 1998, pp. 69 –76.

[104] S. Floyd, "Limited Slow-Start for TCP with Large Congestion Windows," *RFC 3742*, March 2004.

[105] X. Lu, K. Zhang, C. P. Fu, and C. H. Foh, "A sender-side tcp enhancement for startup performance in high-speed long-delay networks," in *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, 2010, pp. 1 –5.

[106] L. Brakmo and L. Peterson, "Tcp vegas: end to end congestion avoidance on a global internet," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 8, pp. 1465 –1480, Oct. 1995.

[107] K. Xu, Y. Tian, and N. Ansari, "TCP-Jersey for wireless IP communications," *IEEE J. Sel. Areas Commun.*, vol. 22, no. 4, pp. 747 – 756, May 2004.

[108] K. Xu and N. Ansari, "Stability and fairness of rate estimation-based AIAD congestion control in TCP," *IEEE Commun. Lett.*, vol. 9, no. 4, pp. 378 – 380, April 2005.

[109] K. Xu, Y. Tian, and N. Ansari, "Improving TCP Performance in Integrated Wireless Communications Networks," *Computer Networks*, vol. 47, no. 2, pp. 219–237, February 2005.

[110] H. Nishiyama, N. Ansari, and N. Kato, "Wireless loss-tolerant congestion control protocol based on dynamic aimd theory," *IEEE Wireless Commun.*, vol. 17, no. 2, pp. 7 –14, April 2010.

[111] S. Floyd, "Highspeed TCP for large congestion window," *IETF RFC 3649*, Dec. 2003.

[112] T. Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks," *SIGCOMM Computer Communication Review*, vol. 33, pp. 83–91, April 2003.

[113] M. Tekala and R. Szabo, "Evaluation of Scalable TCP," in *Proc. 3rd ACS/IEEE International Conference on Computer Systems and Applications*, 2005.

[114] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, Architecture, Algorithms, Performance," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1246 –1259, Dec. 2006.

[115] M. Gerla, B. K. F. Ng, M. Y. Sanadidi, M. Valla, and R. Wang, "TCP Westwood with Adaptive Bandwidth Estimation to Improve Efficiency/Friendliness Tradeoffs," *Computer Communications*, vol. 27, pp. 41–58, 2003.

[116] L. Xu, K. Harfoush, and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," in *Proc. INFOCOM*, vol. 4, March 2004, pp. 2514 – 2524.

[117] S. Ha, I. Rhee, and L. Xu, "CUBIC: a New TCP-Friendly High-Speed TCP Variant," *SIGOPS Operation System Review*, vol. 42, pp. 64–74, July 2008.

[118] J. Brassil, R. McGeer, R. Rajagopalan, P. Sharma, P. Yalagandula, S. Banerjee, D. P. Reed, and S.-J. Lee, "The CHART System: a High-Performance, Fair Transport Architecture Based on Explicit-Rate Signaling," *Operating Systems Review*, vol. 43, pp. 26–35, Jan. 2009.

[119] W. Feng, P. Balaji, C. Baron, L. Bhuyan, and D. Panda, "Performance characterization of a 10-Gigabit Ethernet TOE," in *Proc. 13th Symp. on High Performance Interconnects*, Aug. 2005, pp. 58 – 63.

[120] N. Carpenter, "SMB/CIFS Performance Over WAN Links," http://blogs.technet.com/b/neilcar/archive/2004/10/26/247903.aspx, 2009.

[121] F5 Networks, "F5 WANJet CIFS Acceleration," http://www.f5.com/pdf/white-papers/cifs-wp.pdf.

[122] ——, "F5 WANJet Transparent Data Reduction," http://www.f5.com/pdf/white-papers/wanjet-tdr-wp.pdf.

[123] Makani Networks, "Makani CIFS Acceleration," http://www.makaninetworks.com/register/docs/makani-cifs.pdf.

[124] R. Chakravorty, S. Banerjee, P. Rodriguez, J. Chesterfield, and I. Pratt, "Performance optimizations for wireless wide-area networks: comparative study and experimental evaluation," in *Proc. 10th annual international conference on Mobile computing and networking (MobiCom)*, Philadelphia, PA, USA, 2004, pp. 159–173.

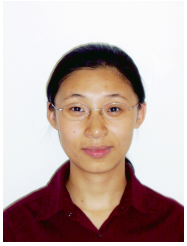
[125] G. Apostolopoulos, V. Peris, and D. Saha, "Transport Layer Security: How much does it really cost?" in *Proc. IEEE International Conference on Computer Communications (INFOCOM'99)*, New York, NY, USA, Mar. 1999, pp. 717 –725.

[126] C. Coarfa, P. Druschel, and D. S. Wallach, "Performance analysis of tls web servers," in *Proc. Network and Distributed System Security Symposium (NDSS'02)*, San Diego, CA, USA, Feb. 2002, pp. 553–558.

[127] K. J. Ma and R. Bartoš, "Analysis of transport optimization techniques," in *Proc. International Conference on Web Services (ICWS)*, Sept. 2006, pp. 611 –620.

[128] D. Boneh and H. Shacham, "Fast Variants of RSA," *RSA Cryptobytes*, vol. 5, no. 1, pp. 1–9, Winter/Spring 2002.

[129] H. Shacham and D. Boneh, "Improving SSL Handshake Performance via Batching," in *Proc. CT-RSA 2001*, D. Naccache, Ed., vol. 2020, 2001, pp. 28–43.
[130] A. Murat Fiskiran and R. Lee, "Fast parallel table lookups to accelerate symmetric-key cryptography," in *Proc. International Conference on Information Technology: Coding and Computing (ITCC)*, 2005, pp. 526 – 531.
[131] V. Gopal, S. Grover, and M. Kounavis, "Fast multiplication techniques for public key cryptography," in *Proc. IEEE Symposium on Computers and Communications (ISCC)*, 2008, pp. 316 –325.
[132] "Speeding Up Secure Web Transactions Using Elliptic Curve Cryptgraphy," in *Proc. 11th Network and Systems Security Symposium*, 2004, pp. 231–239.
[133] W. Chelton and M. Benaissa, "Fast elliptic curve cryptography on fpga," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 2, pp. 198 –205, 2008.
[134] Z. Jiahong, X. Tinggang, and F. Xiangyan, "A Fast Hardware Implementation of Elliptic Curve Cryptography," in *Proc. Information Science and Engineering (ICISE)*, 2009, pp. 1519 –1522.
[135] C. Peng, Q. Zhang, and C. Tang, "Improved tls handshake protocols using identity-based cryptography," in *Proc. International Symposium on Information Engineering and Electronic Commerce (IEEC)*, May 2009, pp. 135 –139.
[136] C. Castelluccia, E. Mykletun, and G. Tsudik, "Improving secure server performance by re-balancing ssl/tls handshakes," in *Proc. ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Taipei, Taiwan, 2006, pp. 26–34.
[137] N. Okamoto, S. Kimura, and Y. Ebihara, "An introduction of compression algorithms into SSL/TLS and proposal of compression algorithms specialized for application," in *Proc. Advanced Information Networking and Applications (AINA)*, 2003, pp. 817 – 820.
[138] S. Hollenbeck, "Transport Layer Security Protocol Compression Methods," *RFC 3749*, May 2004.
[139] R. Friend, "Transport Layer Security (TLS) Protocol Compression Using Lempel-Ziv-Stac (LZS)," *RFC 3943*, November 2004.
[140] K. Jang, S. Han, S. Han, S. Moon, and K. Park, "Accelerating SSL with GPUs," in *Proc. ACM SIGCOMM*, New Delhi, India, 2010, pp. 437–438.
[141] T. Berson, D. Dean, M. Franklin, D. Smetters, and M. Spreitzer, "Cryptography as a Network Service," in *Proc. 7th Network and Systems Security Symposium*, 2001.



Yan Zhang (S'10) received the B.E. and M.E. degrees in electrical engineering from Shandong University, Jinan, Shandong, China, in 2001 and 2004, respectively. She is currently pursuing the Ph.D. degree in the Department of Electrical and Computer Engineering at New Jersey Institute of Technology. Her research interests include automatic modulation classification algorithms, distributed detection in sensor network, congestion control in data center networks, content delivery acceleration over wide area networks, and energy-efficient networking.

Nirwan Ansari (S'78-M'83-SM'94-F'09) received the B.S.E.E. (*summa cum laude* with a perfect GPA) from the New Jersey Institute of Technology (NJIT), Newark, in 1982, the M.S.E.E. degree from University of Michigan, Ann Arbor, in 1983, and the Ph.D degree from Purdue University, West Lafayette, IN, in 1988.

He joined NJIT's Department of Electrical and Computer Engineering as Assistant Professor in 1988, became a tenured Associate Professor in 1993, and has been a Full Professor since 1997. He has also assumed various administrative positions at NJIT. He was Visiting (Chair) Professor at several universities. He authored *Computational Intelligence for Optimization* (Springer, 1997, translated into Chinese in 2000) with E. S. H. Hou, and edited *Neural Networks in Telecommunications* (Springer, 1994) with B. Yuhas. He has also contributed over 350 technical papers, over one third of which were published in widely cited refereed journals/magazines. He has also guest-edited a number of special issues, covering various emerging topics in communications and networking. His current research focuses on various aspects of broadband networks and multimedia communications.

Prof. Ansari has served on the Editorial Board and Advisory Board of eight journals, including as a Senior Technical Editor of the *IEEE Communications Magazine* (2006-2009). He has served the IEEE in various capacities such as Chair of the IEEE North Jersey Communications Society (COMSOC) Chapter, Chair of the IEEE North Jersey Section, Member of the IEEE Region 1 Board of Governors, Chair of the IEEE COMSOC Networking Technical Committee Cluster, Chair of the IEEE COMSOC Technical Committee on Ad Hoc and Sensor Networks, and Chair/Technical Program Committee Chair of several conferences/symposia. Some of his recent recognitions include a 2007 IEEE Leadership Award from the Central Jersey/Princeton Section, NJIT Excellence in Teaching Award in Outstanding Professional Development in 2008, a 2008 IEEE MGA Leadership Award, the 2009 NCE Excellence in Teaching Award, a couple of best paper awards (IC-NIDC 2009 and IEEE GLOBECOM 2010), a 2010 Thomas Alva Edison Patent Award, and designation as an IEEE Communications Society Distinguished Lecturer (2006-2009, two terms).

Mingquan Wu received his Ph.D. in electrical engineering from Michigan State University in 2005. From November 2005 to August 2010, he was a member of technical staff in Thomson Corporate Research. He joined Huawei as a senior researcher in September 2010. His research interests include multimedia reliable transmission over wireless networks, network modeling and resource optimization, ad hoc and overlay network transport protocol design, content delivery acceleration, etc. He has published over 20 referred papers and has more than a dozen pending patents. He has multiple proposals accepted by IEEE 802.11s, IEEE802.11aa and IEEE802.16j standards.

Heather Yu got her Ph.D. in Electrical Engineering from Princeton University in 1998. Currently, she is the Director of the Huawei Media Networking Lab located at Bridgewater, NJ. With the mission of establishing a world class R&D team and leading the key multimedia technology innovations, she led the NJ team successfully accomplished the development of several new media technology research areas and a series of new technology innovations offering competitive edge capabilities and supporting various functionalities for Huawei's products. Before joining Huawei, she was with Panasonic Princeton Lab working on media communication, media processing, media security, and P2P technology research. Since graduated from Princeton, Heather served numerous positions in related associations, such as Chair of the IEEE Multimedia Communications Tech Committee, IEEE Communications Society Strategic Planning Committee member, IEEE Human Centric Communications emerging technology committee chair, Associate Editor in Chief for PPNA journal, AEs of several IEEE journals/magazines, and Conference chair and TPC chair for many conferences in the field. She holds 23 granted US patents and has many in pending. She published 70+ publications, including 4 books, P2P Networking and Applications, Semantic Computing, P2P Handbooks, and Multimedia Security Technologies for Digital Rights Management.