Bandwidth Aggregation in Allied WiFi Routers

Daniel J. Robertson, Craig A. Shue, Krishna K. Venkatasubramanian, Curtis R. Taylor
Worcester Polytechnic Institute
{drob, kven}@wpi.edu, {cshue, crtaylor}@cs.wpi.edu

***Abstract*—Residential and small business customers often have need for high instantaneous download throughput for their Internet transactions which can exceed the throughput provided by the customer's Internet Service Provider (ISP). At the same time, these peak demand periods are relatively brief and may constitute only a few hours a day. During the non-peak periods, these customers have an unused Internet capacity that could be used by others.**

In this work, we propose and evaluate a bandwidth aggregation system for wireless routers. This system allows neighboring users to form bandwidth pools from the users' connections to their ISPs, allowing users to achieve aggregated instantaneous bandwidth. By transparently segmenting large Web requests, our approach avoids changes to the users' machines, ISPs, or remote servers. We implement and evaluate such a system and discuss related issues.

I. Introduction

Internet users have need for high instantaneous download bandwidth for services, such as streaming media. However, to ensure they have the combined capacity to meet the demands from all their customers, Internet Service Providers (ISPs) regularly limit the instantaneous bandwidth available to each customer. As a result, even if the network is highly underutilized, a customer cannot exceed the ISP's limits. This rough-grain restriction can actually exacerbate network congestion during peak demand by slowing connections that otherwise could have completed before peak demand begins. While these restrictions may be suboptimal, they are relatively simple for an ISP to implement and they provide thresholds required for capacity planning.

Fortunately, end-users may be able to alleviate these inefficiencies and achieve higher instantaneous throughput through cooperative bandwidth pools. Unlike traditional bandwidth aggregation techniques, in which a single customer simultaneously uses multiple network connections, our approach would allow multiple autonomous customers to share unused network capacity with neighbors in exchange for additional capacity in the future. Accordingly, a user can achieve higher instantaneous bandwidth if there is spare capacity in the cooperative pool. If there is no additional pool capacity, the user will be limited to the capacity available via the user's direct ISP connection.

To create such cooperative pools, users must have a communication channel that allows them to share their connections with others. In urban and suburban settings, users are often within WiFi range of other users. Accordingly, cooperative WiFi routers may communicate with each other to share capacity.

In this work, we propose and explore using cooperative WiFi routers to aggregate bandwidth across independent ISP customers. In doing so, we make the following contributions:

- **Router aggregation software and protocol:** We create a transparent proxy server that intercepts HTTP requests at a wireless router and, if the server's response indicates a large content stream, the proxy server divides the stream into chunks and issues requests via cooperative routers.
- **Performance evaluation of aggregation approach:** Using a VM environment, we evaluate the approach on a variety of file sizes and chunk sizes to determine the optimal settings. We find that with well chosen settings, our aggregation implementation achieves around 90% of the theoretical throughput maximum.
- **Discussion of obstacles and potential solutions:** Security and liability, particularly non-repudiation, play a significant factor in the practical adoption of the aggregation approach. We discuss these issues and describe approaches that can overcome these concerns.

II. Background and Related Work

Bandwidth aggregation has a rich history in the networking community. While space constraints limit our ability to be comprehensive, we highlight the most relevant background and related work.

Several different layers in the ISO network stack have been used for link aggregation. The IEEE 802.3-2000 standard made the first attempt to unite the existing proprietary link aggregation protocols [1]. The result was the Link Aggregation Control Protocol (LACP), which was officially designated as 802.3ad (and later moved to 802.1ax). LACP uses the datalink layer to perform aggregation within a LAN [2].

Other work has aggregated across different physical and datalink layers. Ramaboli *et al.* present a comprehensive review of the current climate of bandwidth aggregation in heterogeneous wireless networks [3]. They

focus on issues such as packet reordering, which has a significant impact in bandwidth aggregation. In RFC 5236, Jayasumana *et al.* suggest the Reorder Density (RD) and Reorder Buffer-occupancy Density (RBD) metrics for evaluating the degree of packet reordering.

At the application layer, bandwidth aggregation can divide content into chunks and simultaneously transfer these chunks in parallel. While this notion of parallel sockets is similar to our own strategy [4], [5], our approach differs by specifically leveraging different network paths to the same content provider to achieve aggregation. The MuniSockets approach, by Mohamed *et al.* [6], does use separate physical interfaces, it is not responsive to changing network conditions such as delay. Further, the MuniSockets approach requires alterations at both the client and server end-hosts.

At the transport layer, Multipath TCP (MPTCP) has been defined by the IETF in 2011 to support multi-homed systems. A Linux kernel implementation is being explored [7] while Apple has implemented MPTCP in their iOS7 software for use in the Siri service. MPTCP uses a kernel stack to glue together separate TCP flows into a single socket for the application, providing support for any applications already using TCP [8].

Another transport layer protocol, Stream Control Transmission Protocol (SCTP), has multi-homing support build in. The SCTP stack on the end hosts handles packet reordering and enforces congestion control [9]. Although SCTP was not designed around multi-homing specifically, it provides a framework for experimental work. Many SCTP variants achieve bandwidth aggregation, some of which are described in our overview of adaptive bandwidth aggregation solutions.

Network layer bandwidth aggregation solutions provide transparency to upper layers. Unfortunately, they are prone to out of sequence arrivals, which must be handled while consuming as little buffer space as possible. Round-robin packet scheduling works well when packet sizes and transmission rates are homogeneous. However, this is rarely the case in practice and such scheduling could lead to the effective bandwidth of the slowest path. Kim *et al.* proposed a bandwidth aggregation scheme which employs two metrics for scheduling: bandwidth estimation and packet partition scheduling [10]. The former determines the amount of bytes that can safely be transmitted across a link without triggering congestion. The latter decides how packets can be assigned to different paths in order to effectively balance load. A partition counter is assigned to each path, which is used to determine whether the associated path can accommodate a new packet.

Evensen *et al.* introduced a method that uses network stripping, a process which splits traffic over multiple different links, in order to aggregate bandwidth for multi-homed clients [11]. In order to minimize reordering at the client, they employ a smart proxy to buffer out of order deliveries, and delegate packets to different interfaces. The proxy acts as both a scheduler and delay equalizer. It decides which links to send traffic through based off of observed throughput. The delay equalizer is used to mitigate client side reordering, by delaying packet retransmission.

Telefonica has proposed a wireless aggregation solution, BeWifi [12], that uses a mesh-like network to provide aggregation, similar to our approach. However, since the approach is considered proprietary, no technical details on the approach are publicly available.

While each of these approaches have their benefits, they have limitations that hinder their deployment, such as modifications at client and server systems. Instead, we have implemented a cross-layer approach at the user's wireless router that acts as a proxy and uses separate physical connections to aggregate bandwidth. To our knowledge, this approach is unique in the literature.

III. Approach

To provide bandwidth aggregation, we design an approach that allows routers to cooperate. When a router's directly attached user is requesting a resource, that router assumes the *coordinator* role. A coordinator router is ultimately responsible for completing the network transaction. If the coordinator believes the request could be easily parallelized, the coordinator may solicit additional routers to request assistance with the request. These solicited routers, called *helpers*, may optionally choose to provide a portion of their own locally attached bandwidth to aid the coordinator. Since each router will be handling multiple network requests for multiple users, these routers will often act as both a coordinator for its own users and a helper for others simultaneously. However, once a router begins requesting help from other routers to increase its available instantaneous bandwidth, it will naturally reduce the assistance it provides to other routers until it has mets its user's demand. An example of such a network is shown in Figure 1.

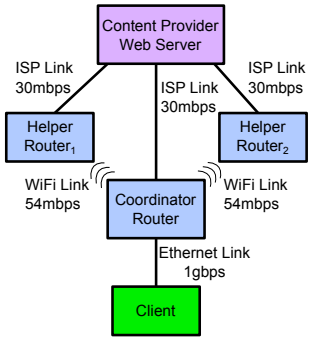

Fig. 1. Example network of cooperative routers.

To initiate a cooperative session, the coordinator broadcasts a request to all neighboring routers with the URL to be retrieved and, ideally, an estimate of the transfer size. The routers receiving the broadcast then independently choose whether to participate in the session or not. If the recipient wishes to participate, it sends an `ACCEPT` response with an estimate of the instantaneous bandwidth that it can provide. At this point, the recipient formally becomes a helper. If the recipient chooses not to participate, it sends a `DECLINE` response with an explanation, such as indicating the router is itself too busy, if the coordinator is untrusted, or if the coordinator has accumulated too high of an assistance debt (i.e., the coordinator has not helped the recipient enough with the recipient's own requests in the past). Further, a helper engaged in a transmission may discontinue its involvement by sending a `DROP` message.

Once the coordinator receives responses from each neighboring router, it knows which routers can help and roughly what bandwidth is available. The coordinator must then segment the request into a series of chunks in order to transfer the file effectively.

While intuitive, our approach faces non-trivial challenges in determining when aggregation will be useful for a given transfer, segmenting requests to achieve optimal performance, and the management of security and liability. We now describe our approach to address each of these challenges in greater detail.

A. Determining When Aggregation is Beneficial

Many Web transactions are small and not worth the coordination overhead associated with the aggregation process. Accordingly, the coordinator must determine when aggregation is worthwhile. Fortunately, many Web server responses will include a HTTP content-length header containing the size of the response in bytes. For responses above a given threshold, the coordinator may abort the on-going Web transfer and request assistance from helper routers. Based on the responses from helpers, the coordinator can divide the request into segments and request helper routers to issue HTTP requests for each of these segments. In particular, the coordinator specifies that each helper router should issue an HTTP request to the indicate URL while including a specified HTTP `Range` header. By issuing non-overlapping ranges, the coordinator can obtain and combine these segments from each router.

The optimal threshold associated with determining when to parallelize will vary based on the latency in communicating with helpers along with the estimated bandwidth and latency of each participant. While it may be challenging to determine the exact optimal value in each circumstance, we explore such a threshold via empirical measurements to help determine potential values.

B. Segmenting Requests for Optimal Performance

A naïve approach for segmenting would be to simply divide a file download into N pieces, where N represents the number of routers involved in the transaction, including the coordinator. The coordinator could then order each helper to provide the indicated range. While straightforward, this approach is unappealing in practice. It may require the coordinating router to buffer large amounts of data. Buffering will occur when segments latter in the file finish downloading before the former segments. In the worst case scenario, the router with the slowest network connection would receive the first segment and the coordinating router would be required to buffer the content from the $2...(N-1)$ segments until the first segment is fully available. On resource constrained routers, this overhead can be significant and essentially infeasible. Accordingly, we consider other methods for dividing chunks between cooperating routers.

As an alternative, we employ a round-robin slicing model where each participating router is assigned a small chunk of data starting at the beginning of the file. The coordinator essentially creates a queue of $\frac{total\_size}{chunk\_size}$ chunks that it will issue as requests. Once a participating router completes an assigned chunk, it receives another chunk in a similar fashion from the coordinator. This approach allows rapid parallelization while the smaller chunk size allows the coordinating router to quickly remove buffered data and transmit it to the user. This reduces memory and storage requirements at the router. There is tradeoff in selecting the chunk size. A smaller chunk size will reduce buffering requirements, but it does so at the cost of additional communication and coordination overhead between participating routers and the coordinator. We explore this tradeoff empirically to determine feasible values.

C. Management of Security and Liability

When a helper router acts on behalf of another party, it may be directed to perform an illegal act, such as downloading contraband or participating in a denial-of-service attack. From a liability perspective, it is essential that the helper router be able to provably demonstrate that it was acting on behalf of another party. Essentially, the helper router wants a non-repudiation guarantee from the coordinating router: the helper can attribute the request to the coordinating router and that coordinating router cannot plausibly dispute the attribution.

To ensure non-repudiation, we use a digital signatures approach using a trusted third party. The third party essentially acts as a notary, allowing the helper to gain an independent witness that the coordinator requested a particular network transaction.

We note that when requesting assistance from other helper routers, the coordinator must sacrifice some con-

fidentiality. In particular, by placing these helpers on the path between the user and the content provider, the coordinator informs the helpers about the content the user is requesting and enables the helpers to obtain a copy of the material being transmitted. Further, if the content is non-public, any credentials required to authenticate to the provider, such as cookie values, must also be shared as part of the transaction. This may be acceptable for popular open video sharing or streaming Web sites, but such disclosure would be inappropriate for any sensitive or non-public content. With server support or proxying services, like the Tor network, users can create an encrypted tunnel to the content provider.

Helper routers may also attempt to modify the communication received from the server to introduce errors or malicious data. However, we can use an independent verification technique to combat this risk. In such a scenario, there are two parties: a prover (the helper router) and a verifier (the coordinator). The helper provides data purportedly from the content provider. The coordinator can then choose a random, small section of bytes within the range of the segment returned by the helper. The coordinator independently downloads the small section of bytes and compares it with the value returned by the helper. If the values differ, the coordinator knows the helper provided false information or that the content provider responded inconsistently. If the helper is detected as providing false information, the coordinator will discontinue using the helper and may refuse any future aid to the helper. Since a malicious peer cannot know the byte range that will be verified by the coordinator, any substantial alterations have a high probability of being detected.

IV. Implementation

To ensure our approach is compatible with routers used in residential and small business deployments, we examined the low-end commercial router offerings and third-party firmware available. In particular, the OpenWRT [13] project was appealing since it can be installed onto a large variety of commercially available routers that have been adopted widely. The firmware provides package management and the installation and execution of executables in lower-level languages, such as C, and higher level languages, like Python. While we kept this deployment target in mind, we created our implementation on a set of traditional Linux machines with the intention of writing software that could fit within the resource constraints of low-end routers and the OpenWRT firmware.

Rather than attempt to perform raw packet operations, such as using a packet capture library and manipulation tool, we instead focused on Web traffic and the use of a proxy server. We used the Twisted Web Framework, an event-driven networking framework for Python [14]. This framework allowed us to add event handlers to segment Web requests for parallel downloads.

To reuse code efficiently, we also used simple Web requests for inter-router communication for the coordinator and helpers. We digitally sign each request using the Python cryptography library, Pycrypto [15]. This library provides RSA cryptographic support, allowing us to create and verify digital signatures. Upon signing the request, the coordinator broadcasts the message to helpers using an `INIT` request. The helpers respond to the coordinator via HTTP with either an `ACCEPT` or `DECLINE` response. We overload this communication into the HTTP protocol using a custom URL format: `[protocol]://[IP]:[port]/[REQUEST TYPE]`, where the protocol is typically HTTP, the IP address reflects the recipient (which is either the helper or coordinator router), the port is a globally-defined constant for this protocol, and the request type corresponds to the `INIT`, `ACCEPT`, `DECLINE`, or `DROP` messages associated with the coordination protocol.

To ensure non-repudiation, the coordinator router sends a copy of the request to the helper to an independent third-party router that acts as a notary. The coordinator transmits the IP addresses of the coordinator and the indicated requestor, along with a copy of the URL and HTTP ranges to be retrieved. The notary adds a timestamp and signs the entire message using its own private key. This notary returns this information to the coordinator, who then passes it to the helper router. Upon verifying the notary's signature, the helper router can then fetch the requested URL while knowing it has cryptographic evidence to minimize liability.

V. Evaluation

We evaluated our approach in a virtualized environment hosted on a server with 12 cores at 2.8GHz and 64GBytes of RAM. We created two guest virtual machines (VMs) running Ubuntu 13.10 and provisioned each with 2GBytes of RAM. Each VM had two network interfaces, one for communication with other routers (simulating a wireless link) and one for communicating with the content provider Web server (simulating a link with an ISP). The ISP link was rate-limited to 10 mbps using the Linux `tc` utility while the wireless link was rate-limited to 54 mbps, which is the limit of the ubiquitous 802.11g standard. To further ensure the rate limiting at the content provider Web server, we also implemented application-layer rate-limiting. We did not rate-limit the transmissions between the coordinating router and its user, emulating a client connected via a high speed link, such as a gigabit Ethernet connection. For testing, we a 40 MByte test file which we divided into 1 MByte chunks at the coordinating router.

The results of this experiment are promising. The total download speed realized by the client of 18 mbps was close to to the theoretical maximum of 20 mbps. When we used a significantly smaller test file of 7 MBytes, the client realized download speed was 17.3 mbps. These results suggest that the small overhead induced by our aggregation techniques is diminished with larger file sizes. In additional trials of other file sizes in this environment, we see that the realized bandwidth of the client is roughly 90% of the theoretical maximum.

A. Impact of Segment Size on Performance

The size of segments allocated to the helper routers has an important role. We empirically examine the impact of the segment size by varying it in our VM environment. Given that the target environment of resource-constrained routers, we set a segment limit to be 2 MBytes since larger chunk sizes would quickly exhaust the router memory. We introduced a third VM router to the previous scenario and set the ISP bandwidth of each at 30mbps leading to a combined bandwidth pool of 30 mbps. This network is shown in Figure 1.

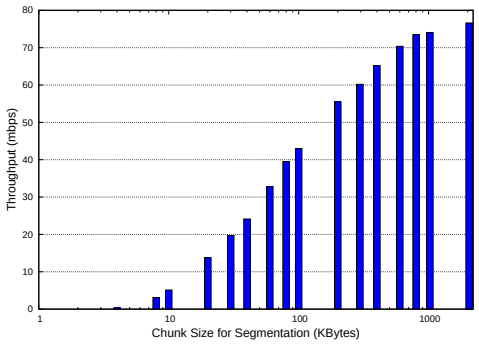

Fig. 2. Effective throughput vs. chunk size among 3 routers with server connections fixed at 30 mbps and 54 mbps wireless links.

We present the results of this experiment in Figure 2. We note that the bandwidth increased dramatically between chunk sizes of 1KByte through 1 MByte, though performance gains moderated after 1 MByte. At the minimum chunk size of 1 KByte, the resulting bandwidth was much lower than using a single router. At that extreme, the realized bandwidth was less than a tenth of a percent of the total theoretical bandwidth available in the pool. At a chunk size of 10 KBytes, the client still received less than 10% of the total theoretical bandwidth. At 100 KByte chunks, the client achieved a combined bandwidth that was roughly 50% of the theoretical offered bandwidth of the participating routers. This milestone is important since many routers have excess RAM capacity that could accommodate 100 KByte chunks. The performance continues to grow dramatically up to chunk sizes around 600 KBytes, which achieves roughly 70mbps. As the chunk sizes near 2 MBytes, the throughput grows modestly to 76.6mbps. Accordingly, we believe the increased buffering requirements beyond 600 KBytes are not offset by sufficient performance gains. For many applications, the optimal chunk range will be between 200 KBytes and 600 KBytes.

VI. Conclusion

In this work, we propose a bandwidth aggregation approach that could be used in commodity wireless routers for residential and small business use. We have created a test implementation of the approach and have empirically shown that it can achieve up to 90% of the theoretical maximum bandwidth available in a pool. We further discussed solutions for liability and security.

References

[1] A. Hendel, "Link aggregation trunking," http://www.ieee802.org/3/trunk_study/tutorial/ahtrunk.pdf, November 1997.

[2] IBM Corporation, "IEEE 802.3ad link aggregation configuration," http://publib.boulder.ibm.com/infocenter/pseries/v5r3/topic/com.ibm.aix.commadmn/doc/commadmndita/ieee8023ad_intro.htm, October 2013.

[3] A. L. Ramaboli, O. E. Falowo, and A. H. Chan, "Bandwidth aggregation in heterogeneous wireless networks: A survey of current approaches and issues," *Journal of Network and Computer Applications*, vol. 35, no. 6, pp. 1674 – 1690, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804512001452

[4] R. Karrer and E. Knightly, "TCP-PARIS: a parallel download protocol for replicas," in *Web Content Caching and Distribution, 2005. WCW 2005. 10th International Workshop on*, Sept 2005, pp. 15–25.

[5] J.-W. Park, J. Kim, and R. Karrer, "TCP-ROME: A transport-layer approach to enhance quality of experience for online media streaming," in *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, June 2008, pp. 249–258.

[6] N. Mohamed, J. Al-Jaroodi, H. Jiang, and D. R. Swanson, "A user-level socket layer over multiple physical network interfaces." in *IASTED PDCS*, 2002, pp. 804–810.

[7] S. Barré, C. Paasch, and O. Bonaventure, "Multipath TCP: from theory to practice," in *NETWORKING 2011*. Springer, 2011, pp. 444–457.

[8] O. Bonaventure, "Decoupling TCP from IP with multi-path TCP," http://multipath-tcp.org/data/MultipathTCP-netsys.pdf, March 2013.

[9] H. Adhari, T. Dreibholz, M. Becke, E. Rathgeb, and M. Tüxen, "Evaluation of concurrent multipath transfer over dissimilar paths," in *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, March 2011, pp. 708–714.

[10] P.-S. Kim, Y. Yoon, and H. Kim, "A packet partition scheduling mechanism for bandwidth aggregation over multiple paths," *Journal of Convergence Information Technology*, vol. 3, no. 4, pp. 325–41, 2008.

[11] K. Evensen, D. Kaspar, P. Engelstad, A. Hansen, C. Griwodz, and P. Halvorsen, "A network-layer proxy for bandwidth aggregation and reduction of IP packet reordering," in *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on*, Oct 2009, pp. 585–592.

[12] *BeWifi*, 2014 (accessed July 1, 2014). [Online]. Available: http://www.tid.es/research/areas/bewifi

[13] "OpenWrt wireless freedom," https://openwrt.org, 2014.

[14] *Twisted Matrix*, 2014 (accessed July 1, 2014). [Online]. Available: http://twistedmatrix.com/trac/

[15] *PyCrypto - The Python Cryptography Toolkit*, 2014 (accessed July 1, 2014). [Online]. Available: https://www.dlitz.net/software/pycrypto/