SAABCOT: Secure Application-Agnostic Bandwidth COnservation Techniques (Invited Paper)

Chad D. Mano
Department of Computer Science
Utah State University
Logan, UT 84322 USA
E-mail: *chad.mano@usu.edu*

David C. Salyers, Qi Liao, Andrew Blaich, Aaron Striegel
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556 USA
Email:{*dsalyers, qliao, ablaich, striegel*}@nd.edu

***Abstract*— High speed modern networks are tasked with moving large amounts of data to diverse groups of interested parties. Often under heavy loads, a significant portion of the data exhibits large amounts of redundancy on short and/or long-term time scales. As a result, a large body of work has emerged offering bandwidth conservation exemplified by the work in caching and multicast. The majority of the techniques that have experienced widespread adoption rely on parsing / reacting to application-specific data. With the advent of simplified end-to-end security, as introduced by IPv6, these techniques will no longer have access to the plaintext data. We present a novel technique for preserving security while allowing in-network devices to identify redundant data flows in order to apply bandwidth conservation techniques. Our communication protocol does not require modifications to existing applications nor does it inflict a significant amount of overhead to the existing network infrastructure.**

I. Introduction

With the rise of multimedia applications, networks are often faced with the challenge of moving large volumes of data to diverse groups of interested parties. Despite significant improvements in network capacity, these volumes of data must pass through constrained upstream links before reaching the high speed core of the Internet. Fortunately, multimedia data often exhibits large degrees of redundancy that can be exploited to improve transfer efficiency.

While bandwidth conservation techniques such as caching [1] and multicast [2] have been proposed and adopted with varying degrees of success, the emergence of end-to-end security scenarios significantly complicate or even preclude the use of various approaches. Specifically, the transition from gateway-oriented VPNs as typically seen with IPv4 to a true end-to-end security relationship envisioned with IPv6 presents numerous issues for bandwidth conservation. As many bandwidth conservation approaches achieve their benefit by acting on plaintext data, the usage of end-to-end tunnels negates the ability of the in-network device to recognize redundant data.

In short, the edge gateway type of environment allows for a de facto sense of trust, i.e. devices on the LAN or in-line with network traffic are trusted. In contrast, trust in an IPv6 environment is solely associated with only the end devices. Thus, in order to offer a benefit, trust must be extended to the bandwidth conservation devices themselves (proxies, caches, etc.) which is non-trivial. Critically, trust extension protocols often offer new opportunities for attack either by virtue of their complexity (DoS or DDoS) or by simply offering a new point to compromise the data flow.

It is this weakness that forms the motivation for this paper: is it possible to allow bandwidth conservation by in-network devices without directly extending trust (i.e. view of the plaintext itself)? To that end, we propose SAABCOT (Secure Application-Agnostic Bandwidth COnservation Techniques), an enhancement to IPsec [3]. SAABCOT incorporates an embedded data-centric key protected by the IPsec session key to enable bandwidth conservation in legacy applications while remaining application-agnostic.

The remainder of the paper is organized as follows. Section II describes related work and further describes the key contributions of SAABCOT. Next, Section III describes the SAABCOT architecture and offers commentary on SAABCOT applications. Finally, Section IV looks at several case studies and Section V offers several summary remarks and comments on our ongoing work.

II. Related Work

While there has been a significant amount of work in the field of bandwidth conservation, research associated with secure bandwidth conservation has been primarily focused on secure group-wise communications [4]. Typically, such works often rely on an assumed multicast infrastructure or the ability to insert an application-layer proxy with modifications provided to the application. Importantly, the issue of deployment, specifically the notion of a required global infrastructure and modifications to the application base have been cited as the primary reasons for multicast deployment falling far short of expectations [2].

In contrast, the work focused on caching [1], has received considerable deployment in the Internet. Applications and pass-through proxies have taken a transparent approach, favoring the immediacy of benefit and deployment, thus dra-

This research was supported by the National Science Foundation through the grant CNS03-47392.

matically accelerating their uptake. Unfortunately, the work in caching and its derivatives presents security risks to the end-to-end security of wide area network communication. Since bandwidth conservation devices remove redundancy by understanding the protocol (proxy-style) or observing the data payload (packet caching), an implicit requirement for their functionality is the ability to process the relevant plaintext of the packet. Put simply, in order to know what to cache, the proxy must understand how to request the data and recognize/respond appropriately. Thus, the application or security model must be modified to create a new security protocol that extends trust to the device in question.

To the best of our knowledge, the only work to consider the impact of security on application-agnostic bandwidth conservation devices is our own work on Trusted Security Devices (TSD) [5]. The TSD work examined how to create an implied group key through extensions of trust in the Xbox Live environment for enabling stealth multicast [6] despite IPsec. The work highlighted a critical weakness in extending trust to additional devices, be those devices proxies or otherwise "trust-worthy" devices, new complex protocols must be developed and devices themselves may now be privy to unencrypted data. The addition of significant complexity lends itself to new avenues for Denial of Service (DoS) or other vulnerabilities while extensions of trust of unencrypted data offers new potential sites to compromise.

This work, SAABCOT, offers several key contributions that differentiate it from the current state-of-the-art work:

- *Lightweight, robust conservation protocols:* SAABCOT offers bandwidth conservation while avoiding the extension of trust in order to minimize data exposure. The minimization of data exposure is critical to avoiding opportunities for adversary Man-In-The-Middle (MITM) attacks or new avenues for Denial of Service (DoS) attacks.
- *Multiple levels of savings:* Previous work has focused on either exclusively saving short-term or long-term redundancy. This work offers a platform to achieve savings in both domains to address the rising needs of streaming data but yet still offer substantial benefit to legacy applications (file transfer, etc.).
- *Simplified deployment :* The work will seek to minimize the deployment impact with existing IPsec implementations to offer simple yet secure installation techniques in a variety of network scenarios.

III. SAABCOT Architecture

In short, SAABCOT provides the ability to offer bandwidth conservation at multiple levels (local, ISP) without complex security interactions nor shared distribution of the plaintext data. This system assumes that bandwidth conservation are likely to be implemented with passive conservation techniques but does not mandate the inclusion of in-network devices. The data sent from each SAABCOT-enabled host is enhanced in order for the existing bandwidth conservation devices to work on the secure encrypted communication.

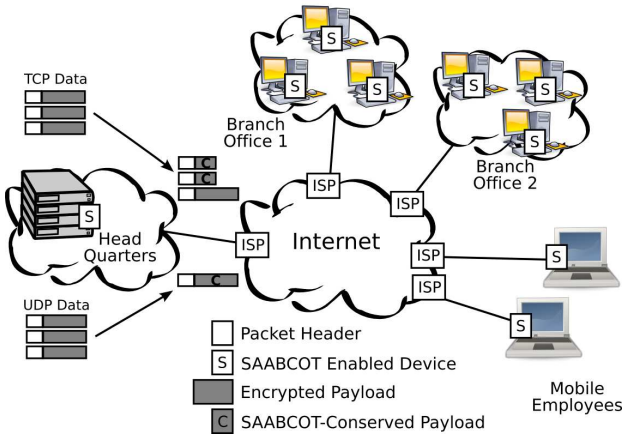

Fig. 1. Example scenario with SAABCOT

Communication begins with an application that transmits data without any special modifications. A SAABCOT-enabled IPsec module (coupled with the IPv6 stack) at the OS level receives the data from the application and encrypts the data appropriately. In-network devices would then be able to recognize redundancy but would not see the plaintext of the data. The actual encryption key for the data would only be available to the end hosts participating in the transmission. This process allows bandwidth conservation to occur, significantly reducing the required bandwidth for transmission of redundant data. TCP data would be tokenized (cached packets are sent using only a token [7], [8]) and UDP packets would be stealth multicast as a single packet [6]. Clients decrypt the data utilizing encrypted information in the SAABCOT header using their end-to-end key in a similar manner to IPsec.

There are three key benefits of the system that bear emphasis. First, in-network devices see only encrypted data and do not see the plaintext. Although it is possible to discern that the same data is being sent to multiple hosts, the content of the data is kept secret. Furthermore, the ability to even detect that the same data is being detected would require a compromise of a device on the communication path, a non-trivial undertaking that could easily inflict far worse damages (DoS, MITM, etc.).

Second, the applications themselves do not need to be modified. Similar to IPsec, SAABCOT is a drop-in module that operates in the OS, thus providing a normal socket-based interface to the application. Unlike proxy-based solutions or multicast solutions, SAABCOT will work with legacy applications without application modifications, thus offering an immediate and tangible benefit.

Third, SAABCOT need only be enabled at the source and client. In-network support is not required and simple backwards-compatible extensions to the initial IPsec handshake would allow for SAABCOT probing. Hence, SAABCOT will not mandate changes to the existing network, thus further simplifying deployment.

We now follow up this general description of the SAABCOT architecture with details of the communication setup and encryption protocols.

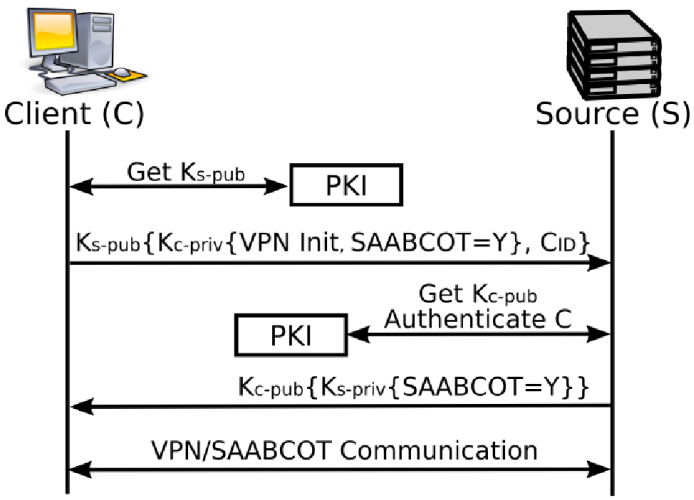

Fig. 2. SAABCOT communication initialization

A. SAABCOT Initialization

For a VPN implementation using the properties of IPsec, the following information is known:

- Each end host has a public and private key (ex. K_{S1-PUB}, $K_{S1-PRIV}$). The public key is known to all and available via a PKI (Public Key Infrastructure) system. The private key is known to only the end host and is used for signing the data.
- Symmetric session keys are used for data exchange between client and server systems ($K_{SYM-S-C_i}$).
- The ability to decrypt a message with the symmetric key is derived from the end-to-end trust, i.e. the only way to receive a symmetric key is if the public/private key relationships are valid. Hence, the internal contents of the message from the server to client C_1 is only readable by C_1.

The goal of SAABCOT operation is to transmit the data payload such that redundancy can be recognized by the in-network devices but yet not visible in terms of plaintext. Importantly, only the end host will have the ability to extract how to decrypt the text from fields encrypted by the end-to-end VPN session key in the SAABCOT header.

The initial setup for SAABCOT occurs during the communication exchange which authenticates users and establishes the symmetric session key described. The only modification to this phase is for the purpose of determining if end client systems can support SAABCOT. If a client does not denote its support for SAABCOT, normal VPN operations are performed with bandwidth conservation not being realized. A simple two-step process is used to establish a SAABCOT communication session as illustrated in Figure 2.

Step 1: Client initiates key exchange The client (C_i) validates the public key of S (ex. the data source at the main office) from the assumed PKI. The client then sends a request to S to initiate the VPN and appends a SAABCOT probe indicating that it is capable of a SAABCOT communication session.

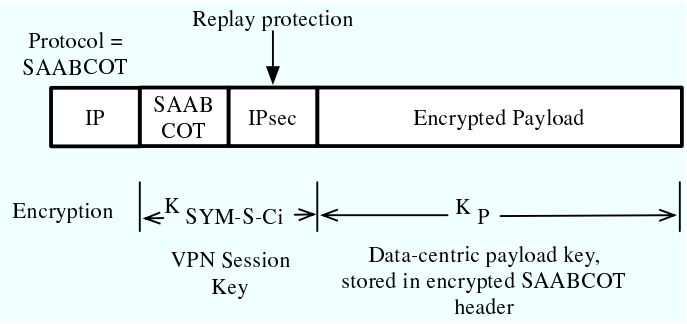

Fig. 3. SAABCOT Data Packet

Step 2: Source receives request, responds to client The source for the data (S) receives the request and validates the public key of C_i via the PKI. S responds and denotes whether or not it supports SAABCOT in the reply. A symmetric session key for SAABCOT (K_{SB}) is also sent from the source to the client. The remainder of the key exchange proceeds as normal.

After the key exchange procedure is completed, both the client and the source are aware that SAABCOT support will be either enabled or disabled for this connection. Similar techniques for expanding IPsec would be used in the case of the standard Diffie-Hellman key exchange versus the above public/private key exchange. Regardless of whether or not SAABCOT-enabled bandwidth conservation devices are present in the network, information exchange can still occur. Thus, even if a SAABCOT device is attacked or eliminated, a Denial of Service will not occur. Rather, a network without SAABCOT-enabled conservation devices will simply not experience any bandwidth savings from SAABCOT. Note that this is a critical difference versus the extension of trust where IPsec tunnels terminate at a bandwidth conservation device.

B. SAABCOT Communication

Figure 3 shows an example SAABCOT data packet. The primary change versus IPsec involves two modifications and one insertion. First, the IP next header field (protocol field in IPv4) is changed from IPsec to SAABCOT to denote how the packet should be handled. Second, the SAABCOT header is inserted as a shim after the IP header and before the IPsec header. Third, the encrypted data payload is encrypted using a key specified by SAABCOT (K_P) of which the new key is protected by the IPsec symmetric session key. Informally, the key relationship can be stated as follows:

$$ESP = E_{K_P}(P)$$

where ESP is the newly encrypted payload, K_P is utilized by SAABCOT to encrypt the payload, and P is the original payload. The encryption key for the payload is included in the SAABCOT protocol header which in turn is encrypted in the following manner:

$$KeyInfo = E_{K_{SB-S-Ci}}(K_P)$$

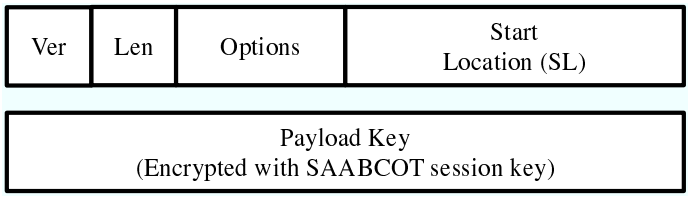

Fig. 4. SAABCOT Packet Header

where $K_{SB-S-Ci}$ is the symmetric session key for SAABCOT between the server (S) and the client (C_i) .

The actual payload key (K_P) is selected by the server on transmission. Selection of K_P from the list is chosen by transforming a secure hash of the unencrypted application-layer payload into a key of the same key length as the IPsec session key:

$$K_P = f(SHash(P, K_H))$$

where $f(x)$ is the transformation function, SHash computes the secure hash, K_H is the server-side key for the secure hash, and P is the payload. At a minimum, the keyed input to the secure hash function should change similar to the IPsec session key. For redundant data with close temporal proximity, the key itself can be shifted quite regularly without any impact on performance as the redundancy windows are quite small. Hence, one could provide backwards secrecy by computing a new payload key K_P on the detection of unique UDP payloads (determined by the secure hash). Conversely, frequent rekeying of data with long term redundancy (ex. TCP) represents a tradeoff of performance versus key exposure as a change of the mapping for K_P would invalidate downstream cached content.

The SAABCOT header (Figure 4) contains 5 key fields:

- *Version / Length:* Similar to original IP header specification
- *Options:* Flags to customize SAABCOT operation
- *Start Location (SL):* The start location of the potentially redundant data as an offset after the IPsec header. Data after the start location is assumed to be encrypted with K_P while data before SL is assumed to be encrypted with $K_{SYM-S-Ci}$
- *Payload Key :* The included key K_P represents the mechanism to decrypt the payload at the end client. The key is encrypted using the end-to-end session key ($K_{SYM-S-Ci}$) as outlined earlier.

To decrypt the message, the end client first decrypts the $KeyInfo$ header using the SAABCOT symmetric key for the specific security relationship ($K_{SB-S-Ci}$) and then uses the resulting key (K_P) to decrypt the packet payload. Note that replay protection is preserved as the original IPsec header is still present and protected by the original IPsec symmetric key. Moreover, it is important to note that the relative security strength to decrypt the actual payload is preserved. While an adversary can potentially determine that the same information is being sent to multiple clients (already somewhat easily inferred even with separate keys by watching the IP size field and temporal relationships), the adversary does not gain any sort of advantage in cracking the encrypted text provided that the key list (kept and generated only server side) is regenerated aggressively. Moreover, we believe there are numerous cases where the lack of strict confidentiality (i.e. cases of identical data may be identified) is not critical to the security of the data flow (distributed backups, secure file sharing, etc.). *In cases where absolute confidentiality trumps efficiency (ex. credit card application response), SAABCOT should simply not be enabled.*

C. SAABCOT Key Strength

Consider the strength of SAABCOT examined more formally. Suppose that an adversary Eve (E) wishes to decrypt the messages going to Alice (A). In the worst case, consider that the E has already been receiving the data legitimately from the server and has saved M keys up to this point ($K_{P1}, K_{P2}, \ldots, K_{PM}$). Furthermore, E is listening to A over an unsecured wireless channel and is able to observe all traffic to A.

In the simplest case, both hosts (A and E) request the same data. In such a case, E can tell that A received the same data. As E had the data legitimately, the only benefit to E is knowing that A also has the same data. The observance of data would be similar to that of various multicast group keying schemes which is not unreasonable given the savings offered by SAABCOT. However, we do note that this behavior is distinct from the per-host keying of IPsec.

With regards to an intelligent attack that consumes less than the cost of a brute force attack, E can attempt a fast guess through the gathered M keys hoping for a quick match. However, provided that collision is unlikely (a reasonable assumption given the relatively small payloads of packets), E will be likely forced to examine the key space of K_P. As K_P has the same key space as the IPsec session key $K_{SYM-S-A}$ and SAABCOT session key (K_{SB-S-A}), E cannot expedite the process aside from the list of known M keys. Furthermore, E is thwarted from short circuiting the key space search of the secure hash (birthday attacks, etc.) as E does not know the input key to the secure hash function. Hence, we argue provided that rekeying is used appropriately (IPsec session rekeying interval), M will not grow large enough to create significant issues.

D. SAABCOT Bandwidth Conservation

From the perspective of the in-network bandwidth conservation device, the device would simply operate on SAABCOT packets, forwarding all other packets without modfication. As the in-network device can now see redundancy, bandwidth conservation techniques can now be applied. The device caches or multicasts/broadcasts the encrypted content to the downstream device, saving considerable bandwidth as either redundant packets are eliminated [6] or tokenized [7], [8]. UDP streaming would be conserved using stealth multicast [6] while TCP transfers would be conserved using either partial or whole packet caching [7], [8].

For example, consider the case where the message "REDUNDANT" is sent to each of two end clients. For demonstration purposes, we use a Caesar cipher with the following symmetric keys for C_1, C_2:

$$K_{SYM1} = \text{“}B\text{”}, K_{SYM2} = \text{“}C\text{”}$$

Without SAABCOT, a bandwidth conservation device would see payloads of:

$$|IPsec|\text{“}SFEVOEBOU\text{”}$$

$$|IPsec|\text{“}TGFWPFCPV\text{”}$$

and hence could not offer any savings. In contrast, with SAABCOT, the device would see messages of:

$$\text{“}(E)|IPsec|[UHGXQGDQW]\text{”}$$

$$\text{“}(F)|IPsec|[UHGXQGDQW]\text{”}$$

where () contains the key index encrypted by the session key, $|IPsec|$ is the original IPsec header encrypted by the IPsec session key, and [] contains the encrypted payload. In this case, a common key $K_P = 3 =$ "D" is included in the SAABCOT header, encrypted by each of the symmetric keys for the clients.

This enables in-network conservation devices to see identical payloads without understanding the meaning of the payload. Hence, bandwidth conservation could occur from the upstream SAABCOT device to the downstream SAABCOT device. For example, a single token of D might be used to represent "UHGXQGDQW" with the SAABCOT header and the token transmitted over the bottleneck links (messages become (E)[D] (F)[D]). At the downstream SAABCOT device, the message is decompressed to send (E)[UHGXQGDQW], (F)[UHGXQGDQW] onwards to the clients.

When the client receives the packet, it would use its symmetric key $K_{SYM-S-Ci}$ for the VPN session to decrypt the message. For example, C_1 would decrypt $E-B = 4-1 = 3 = D$ and apply the derived key to the message:

$$\text{“}UHGXQGDQW\text{”} - 3 = \text{“}REDUNDANT\text{”}$$

C_2 would arrive at the payload key in a similar manner. Note that despite the fact that the payload is the same, there are no hints provided as to what the actual payload key is. In fact, if an adversary could derive K_P from the two messages, the cryptosystem itself would have to possess a fundamental weakness. Furthermore, conservation could be probabilistically reduced in favor of secrecy by probabilistically embedding a random key in the SAABCOT header for selected hosts.

E. SAABCOT Overhead / Security

In terms of overhead, the overhead of SAABCOT is relatively lightweight. In the worst case where the payload key is included in the packet and authentication from the source is desired (digital signature), the overhead is 28 bytes / packet, slightly more than the IP or TCP headers themselves but less than IPsec.

While 28 bytes may seem significant in light of the emphasis on bandwidth conservation, consider the savings that SAABCOT would introduce. In the worst case, the 28 byte header represents an overhead of 1.8% of an Ethernet-style MTU (1500 bytes). Thus, if only 1.8% of the application output traffic is redundant, SAABCOT can offer a break-even savings. Analysis of the University of Notre Dame Internet Gateway shows levels of redundancy ranging from 3% to 20%. Moreover, Chesire et al. noted in [9] that peak periods of redundancy tend to occur at peak traffic times, i.e. when bandwidth conservation is needed most. More recent studies in [10] have noted an explosion in end-user as true high-speed connections (fiber to the home) are delivered.

Finally, we revisit the basic security issues with SAABCOT:

- *Replay:* Replay protection is provided by the IPsec header. The IPsec header is encrypted as normal with the IPsec session key ($K_{SYM-S-Ci}$). Other properties from IPsec (authentication, etc.) are enabled as well through the inclusion of the IPsec header.
- *Collusion:* As the selection of K_P is data-centric, memorization or sharing of individual K_P keys offers little benefit. Provided the K_P space is vast enough to minimize hash collisions and the server aggressively refactors keys, collusion (key sharing) is of minimal concern. Moreover, simple application-layer mechanisms such as those noted in [8] could be employed to mark data as non-redundant whereby none of the packet is encrypted using the in-packet SAABCOT key. The primary tradeoff with regards to key refactorization comes with caching of long-term redundant data that would miss out temporarily on bandwidth savings.
- *Redundancy Identification:* A separate but valid issue is the fact that redundancy between different connections is now visible (i.e. A and B both received the same packets). Hence as noted earlier, SAABCOT may not be ideal for all data. Rather, SAABCOT offers a tradeoff of bandwidth savings / bottleneck reductions versus truly independent end-to-end security. We believe that the savings are significant enough to merit application in many scenarios as the presence of IPsec becomes more prevalent in the network.

IV. Case Studies

To illustrate the performance benefits of SAABCOT, we illustrate three example cases, namely videoconferencing with VPN interconnections, batches of grid computing jobs retrieving data securely, and code check-in via CVS through ssh.

A. Case 1: VPN Videoconferencing

For the first case, consider the case of corporate videoconference involving a main office (MO) and multiple branch offices (BO_1, BO_2, BO_3). The communication is many-to-many style with each site both sending and receiving audio/video information. In addition, several mobile nodes (MN_1, MN_2) are listening to the overall communications for traveling employees to listen in. Each entity is connected by IPsec forming a VPN between all parties.

For the traditional unicast model, each party involved in the communication introduces a linearly increasing cost for outbound communications. Hence, the above example would replicate individual packets 5 times for each of the receivers. Although approaches such as Application Layer Mutlciast (ALM) [11] could reduce the overall bandwidth, the distance between various receivers (ex. between different BOs) would likely deliver unsatisfactory QoS performance. Alternatively, techniques such as Automatic Multicast Tunneling (AMT), if offered, could improve performance as well provided that significant multicast deployment existed and that the VPN gateway acted on behalf of the clients.

In contrast, SAABCOT streamlines the process significantly. At worst case where no bandwidth conservation devices exist, SAABCOT offers near unicast performance. If a stealth multicast device [6] is present, the redundant bandwidth (repeat communications) are conserved over the bottleneck links without modifications to the application. Moreover, aggressive re-keying could be applied to provide backward secrecy for clients leaving the virtual session.

B. Case 2: Grid Computing

For the second case, consider a batch of computing jobs drawing upon the same data to be distributed to a WAN-scale or campus-wide compute grid. Examples of such environments include the Open Science Grid (OSG), Globus, Folding@Home [12], SETI@Home [13], and others [14], [15]. In those cases, the data can be copied in a secure manner from the source site in either a push (data staging) or pull (job retrieves data) manner. The mechanisms for I/O include tools such as GridFTP, scp, and AFS.

Under the traditional unicast model, each job staged will incur a linearly increasing penalty for distribution. Thus, for examples such as Folding@Home or SETI@Home where data is distributed to multiple clients for reliability/verification purposes, significant bandwidth can be considered redundant. The use of object-style caching [16] could offer a benefit if requests for a site are directed through a proxy. However, such cases are extremely rare with data retrieval often directed by the submitter of the computation job.

In contrast, consider the usage of SAABCOT. With SAABCOT, data across both short timescales and long timescales can be conserved. If data is pushed in a large batch (pre-staging), the repetitive data can be consolidated via stealth multicast and the primary bottleneck link significantly improved. If data is pulled from the hosts, the usage of packet caching can alleviate significant repetition of the data. The notion of how to preemptively cache packets of data is an interesting topic for future research [17].

C. Case 3: Code Check-in

Finally, consider a group of contract programmers checking in code revisions to a central CVS repository. Similarly, consider uploading executables or code for compilation to remote testing machines such as for PlanetLab [18] or Emulab [19]. While the majority of the code itself is not likely to be unique, the repository must still be validated to isolate code changes. Similarly, the code for remote testing often contains significant repetition when copied across the network.

Similar to the earlier cases, SAABCOT can offer a significant improvement depending on the volume of updates sent by the programmer. While most version control attempts to minimize the total transfer of data, each file change could be minimized even further in terms of aggregate transfer cost. The case of remote testing offers the greatest benefit as aggressive packet caching could create the illusion of LAN-like performance when copying files despite operating across the WAN.

V. Summary and Future Directions

In summary, SAABCOT meets an important need for application-agnostic bandwidth conservation in a simple, robust, and secure in an IPv6 environment. Through its coupling with IPsec, SAABCOT offers similarly strong encryption strength but yet allows multiple levels for bandwidth conservation at both the local and ISP level. Moreover, SAABCOT is extremely lightweight, incurring little overhead in terms of the initial negotiation and the per-packet overhead. Thus, we feel that SAABCOT offers an interesting platform for future bandwidth conservation in an IPv6 Internet.

Our current work includes modifications to OpenSSH (ssh, scp) and FreeS/WAN (open source IPsec for Linux) to offer SAABCOT functionality. Beyond SAABCOT, we note several research issues that merit further attention:

- *Reverse Conservation:* In the initial conception of SAABCOT and bandwidth conservation, considerable focus is placed on the forward path or significant producers of data. An interesting item is to consider how much benefit could be gained from examining conservation on the reverse path. Although various works have examined packet aggregation [20], there may be opportunities with initial fetching and data retrieval for bandwidth conservation.
- *Sharing Checksum Information:* In order to derive K_P, SAABCOT employs a transformation function from the secure hash of the payload information. Should this data be shared with conservation devices in the network to help expedite identification of redundant data? Similarly, should data be proactively split [8] to accelerate redundancy computations? Finally, does the block-wise nature of most strong cryptographic schemes negate the usage of partial caching schemes [21] that dynamically detect the boundaries of redundant content?

- *DDoS Potential:* Does bandwidth conservation offer new opportunities for DDoS attacks? As the attacker may no longer be subject to their local bottleneck, bandwidth conservation offers the opportunity for amplification of an attack. Should the rate of conservation be limited on a host-wise or checksum basis?
- *TCP and Bandwidth Conservation:* If one is being a good netizen and using IPsec with bandwidth conservation (identity via IPsec, savings via bandwidth conservation), do the normal rules of TCP self-clocking apply? For instance, given an effective packet cache, packets will nominally be tokenized from their MTU size to roughly bigger than the minimum packet size? Should this bandwidth savings translate into additional bandwidth or reliability or both?

REFERENCES

[1] J. Wang, "A survey of Web caching schemes for the Internet," *ACM Computer Communication Review*, vol. 25, no. 9, pp. 36–46, 1999.

[2] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for IP multicast service and architecture," *IEEE Network*, pp. 78–89, Jan./Feb. 2000.

[3] S. Kent and R. Atkinson, "IP Encapsulating Security Payload (ESP)," *IETF RFC 2406*, Nov. 1998.

[4] S. Setia, S. Zhu, and S. Jajodia, "A comparative performance analysis of reliable group rekey transport protocols for secure multicast," *Performance Evaluation*, vol. 49, pp. 21–41, 2002.

[5] C. Mano and A. Striegel, "Trusted security devices for bandwidth conservation in IPSec environments," in *Proc. of IFIP Networking*, Waterloo, Canada, May 2005, pp. 166–177.

[6] D. Salyers and A. Striegel, "A novel approach to transparent bandwidth conservation," in *Proc. of IFIP Networking*, Waterloo, Canada, May 2005.

[7] J. Santos and D. Wetherall, "Increasing effective link bandwidth by suppressing replicated data," in *Proceedings of the USENIX Annual Technical Conference*, New Orleans, Louisiana, June 1998.

[8] X. Li, D. Salyers, and A. Striegel, "Improving packet cache scalability through the concept of an explicit end of data marker," in *Proc. of IEEE HotWeb*, Boston, MA, Nov. 2006.

[9] M. Chesire, A. Wolman, G. Voelker, and H. Levy, "Measurement and analysis of a streaming media workload," in *Proc. of the 3rd USENIX Symposium on Internet Technologies and Systems*, Mar. 2001.

[10] K. Cho, K. Fukuda, H. Esaki, and A. Kato, "The impact and implications of the growth in residential user-to-user traffic," in *Proc. of ACM SIGCOMM*, Pisa, Italy, Sept. 2006, pp. 207–218.

[11] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Networking Support for Multicast*, Oct. 2002.

[12] "Folding@home project," available at folding.stanford.edu.

[13] "SETI@home project," available at setiathome.ssl.berkeley.edu.

[14] K. K. et. al, "Computational grids in action: The national fusion collaboratory," *Future Generation Computer Systems*, vol. 18, no. 8, pp. 1005–1015, Oct. 2002.

[15] M. R. et. al, "The astrophysics simulation collaboratory: A science portal enabling community software development," *Cluster Computing*, vol. 5, pp. 297–304, 2002.

[16] D. Wessels, "The squid internet object cache." 1997, available at http://squid.nlanr.net/Squid/.

[17] T. Malik, R. Burns, and A. Chaudhury, "Bypass caching: Making scientific databases good network citizens," in *21st International Conference on Data Engineering (ICDE)*, 2005, pp. 94–105.

[18] A. C. Bavier, N. Feamster, M. Huang, L. L. Peterson, and J. Rexford, "In vini veritas: realistic and controlled network experimentation," in *ACM SIGCOMM*, Pisa, Italy, 2006, pp. 3–14.

[19] E. Eide, L. Stoller, and J. Lepreau, "An experimentation workbench for replayable networking research," in *Fourth USENIX Symposium on Networked Systems Design and Implementation (NSDI 2007)*, Cambridge, MA, Apr. 2007.

[20] B. Badrinath and P. Sudame, "Gathercast: The design and implementation of a programmable aggregation mechanism for the internet," in *Proc. of IEEE Int'l Conf. on Computer Communications and Networks (ICCCN)*, Oct. 2000.

[21] N. T. Spring and D. Wetherall, "A protocol independent technique for eliminating redundant network traffic," in *Proc. of the 2000 ACM SIGCOMM Conference*, Stockholm, Sweden, Aug. 2000.