Dynamic Virtual Private Networks

Ohad Rodeh, Ken Birman, Mark Hayden, Danny Dolev
{orodeh,dolev}@cs.huji.ac.il, {ken,hayden}@cs.cornell.edu *

Abstract

We extend traditional Virtual Private Networks (VPNs) with fault-tolerance and dynamic membership properties, defining a Dynamic Virtual Private Network (DVPN). We require no new hardware and make no special assumptions about line security. An implementation exhibits low overheard, provides guarantees of authenticity and confidentiality to any IP application running over the virtual network. Our system is lightweight, allowing the use of multiple fine-grained VPNs. Instead of using many point-to-point secure connections to bridge insecure communication paths we share a single symmetric encryption key throughout the VPN. This permits tight control of the VPN membership and fast dynamic membership change.

Since we lower the cost of a single DVPN, we propose using multiple DVPNs to implement fine grained security. By enforcing policies over communication between DVPNs, our scheme supports multilevel security.

1 Introduction

We say that a Virtual Private Network (VPN) exists between a set of machines if they can communicate over a public infrastructure with the illusion that the communications network has been dedicated for their exclusive use. Existing VPNs provide an attractive solution to network problems[1]. VPNs are transparent (mostly) to the applications running inside them and they offer robust protection. However, they are limited in certain ways:

- Typically, fault tolerance is not handled well. Servers represent single points of failure. For example, the native NT 4.0 firewall relies on a single Kerberos server to authenticate clients. Should this server crash, clients will be denied service.
- New firewalls and IP subnets cannot be dynamically added to a VPN, for example see [18].
- VPNs offer protection to trusted machines through a variety of techniques: IP filtering, encryption, software proxies, virus checking, etc.. This bundles a lot of functionality in one package. Ideally, the user should be able to choose precisely what he needs.

*The authors were supported in part by DARPA ITO under ARPA/ONR contract N0014-96-1-10014 and ARPA/RADC contract F30602-95-1-0047, in part by grants from Microsoft and AT&T, and by the Israeli Ministry of Science grant number 032-7892.

[1]Currently, most commercial VPNs do not address QoS or denial service, thus, full network dedication is not provided.

- Secure point-to-point connections are used to connect firewalls. This would be problematic if the set of firewalls were dynamic. For example, assume two sets of firewalls S_1 of size n and S_2 of size m; $O(n^2)$ connections connect S_1 and $O(m^2)$ connect S_2. To fully connect the sets together (for example, after a network partition) $n * m$ connections must be set up. Similarly, If m are firewalls are removed from the VPN then $(n - m) * m$ connections must be torn down.

We propose *Dynamic Virtual Private Networks* as the next step in system security. A DVPN has the following properties, extending traditional VPNs:

- Decentralized management(fault tolerance): The DVPN does not have a single point of management. Should the DVPN split into several connected components each component is able to manage itself as well as retain capability of remerging with other components when network connectivity is reestablished.

- Scalable dynamic membership: machines may dynamically join and leave the DVPN. When trusted machines join the DVPN they automatically receive up-to-date security keys and other relevant information.

- Lightweight: A DVPN does not consume much machine resources, this allows new uses for DVPNs: running several DVPNs over a single machine, implementing multilevel security and more.

- Technology: We propose using a shared secret key to encrypt communication throughout the DVPN. We use a rigorous mechanism for securing the key's freshness and security such that only trusted and authenticated members receive it. The advantage over secure point-to-point connections is easier handling of dynamic addition and removal of firewalls. When machines are added to the DVPN, only the key need be passed. when a machine ceases to be trusted the DVPN key is simply switched — effectively removing it from the trusted network.

Since a DVPN incurs little overhead we propose using multiple overlapping DVPNs to implement fine grained security. For example, in Figure 1 we depict a company in which the CEO (C) is in the center, the manager group includes C, Ms, and Mp, the workers are Ws, Wp1, and Wp2. The managers form a group M in which they handle their discussions. There are also the Production, and Sales groups which use their private DVPNs for discussions. The CEO is naturally a member of all these groups. There is also a group that encompasses the whole company. DVPNs are used to confine information to authorized machines. Information should not flow from the managers to the workers; to capture such restrictions we propose using an IP filter to enforce a policy for intra-DVPN communication. Such a filter, installed on all the machines in the company, will enforce a policy such as in table 1.

Such filtering can be performed efficiently, and it will prevent information from leaking from machines in M to less secure machines. When can a machine m be a member of DVPNs $D1$ and $D2$? When it is secure enough for both. To make this formal we attach a security level to each DVPN and each machine. Security levels are elements of a grid where the maximum and minimum operations are defined. Thus, a m may be a member of $D1$ and $D2$ only if its security level is greater or equal to $\max(D1, D2)$.

DVPN	M	S	P
M	T	F	F
S	T	T	F
P	T	F	T

Table 1: Example of a policy. Information may flow from M to M, from S to M and S, and from P to P and M.

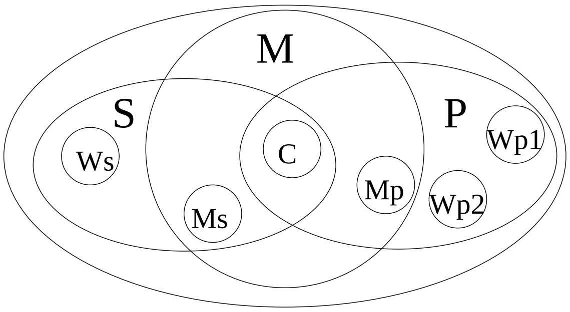

Figure 1: The DVPN constellation in the company. C is the CEO; Mp and Ms are the lower level managers; Ws,Wp1, and Wp2 are the workers. The groups are P for Production, S for Sales, and M for Management.

Processes in a DVPN may need to use insecure services outside the protection scope of the DVPN. For example, the NFS server may not reside inside the DVPN. We propose encrypting all data stored on the NFS with a key known to all members of the DVPN. We do not explore this further in the paper except to note that using such unprotected services securely is formidable challenge.

We have built a system implementing a single DVPN, and have also dabbled with multiple DVPNs (though this has not be made freely available yet). Our work was performed on Linux, but should easily port to NT (on which our software could be used to control the built-in NT VPN technology), Solaris, or other systems. Although we made some modifications to the operating system, these are so standard (a new device driver, some changes to the routing table, a packet filter) that they should be possible on any OS.

In the rest of the paper we describe our DVPN system, its performance and its relation to current commercial VPNs. Section 2 discusses today's VPN architectures, Section 3 briefly describes related work, and Section 4 describes the model and the assumptions that we make. Section 5 describes the security architecture that we are proposing, the following sections describe its different components, and scalability. Section 11 discusses future work and Section 12 concludes. The Appendix includes performance figures, showing low latency and high throughput for our system, and a detailed bibliography.

2 A standard VPN

This section describes a standard VPN. Normally, multiple VPNs are not used to secure an enterprise. Rather, a company's machines are encapsulated within a single VPN to protect them from malicious outsiders.

To build a VPN, the system administrator divides the user group into trusted and untrusted users. A machine is trusted if only trusted users can access it; otherwise, it is untrusted. An IP network is said to be secure if all the machines it contains are trusted. Thus, the global set of machines to be protected is partitioned into a disjoint set of secure IP networks, and distant clients.

The private IP subnets are sealed from the world by *firewalls*. These machines are interpose on all communication paths between the trusted IP network and the external, untrusted, environment. The firewall employs a mixture of proxy mechanisms and packet filtering to protect the inside machines from intruders and also restricts access to the outside. For simplicity, henceforth we focus on VPNs running all their services on the firewalls. This is important, as we shall later see, since a DVPN essentially runs a lightweight firewall on all its members.

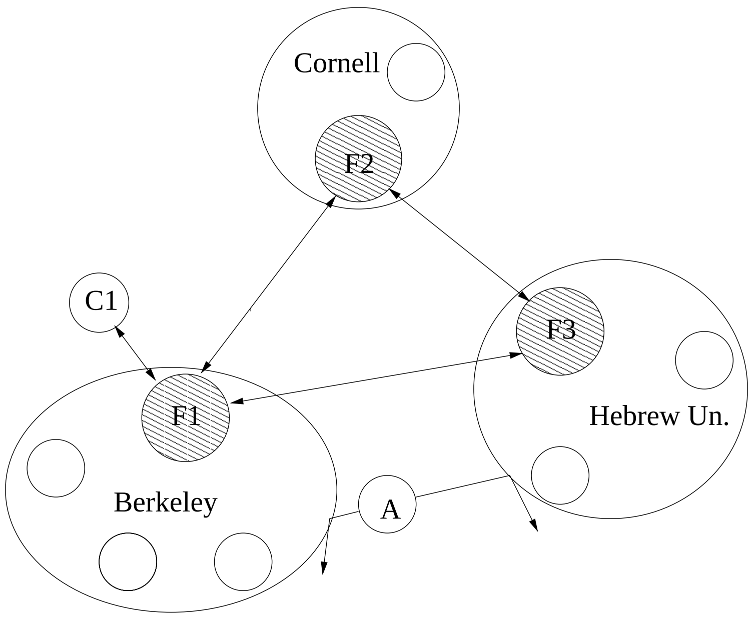

Figure 2: A typical VPN configuration.

In Figure 2 an example of a set of machines in a VPN is shown. There are three trusted IP subnets, at Cornell, Berkeley, and the Hebrew universities, with firewalls at machines $F1, F2, F3$ and a trusted distant client $C1$. The adversary A is trying, illegally, to enter the VPN not through the firewall — his access is denied.

The firewalls are connected by a mesh of secure point-to-point links. To avoid the cost of leased lines, such connections are run over the Internet, using techniques such as PPTP [19] and IP_tunneling. Messages are encrypted and signed to ensure privacy and authenticity. Distant clients are connected to the VPN through one of the firewalls, using a secure connection.

VPNs are useful in many settings:

- Virtual Organizations. The VPN connects hosts belonging to a single institution/company which is geographically spread over several distinct subnets. The organization thus avoids the need to lease dedicated lines, but has protection against Internet cyber-espionage.

- Connecting distant clients to a company's network. Increasingly, companies provide goods and services to their clients over the Internet, granting privileged access denied to non-clients. VPN solutions permit the company to define client networks such that only legitimate users can access network resources, and protecting valuable data against attack.
- Military and intelligence applications. A VPN creates a security containment zone, in which security policies limit information flow both into and out of the zone.
- So-called Community Health Information Networks (CHINs) are being developed in many states to link primary-care physicians and Health Management Organizations (HMOs) with local hospitals and specialized care providers. These networks may be connected directly to medical devices such as IV drips, and will carry sensitive data for which isolation and security are clearly needed.

The usual technology for ensuring the security of a point-to-point connection involves symmetric digital encryption and signature. Data is encrypted to ensure its privacy; it is signed to ensure its authenticity. Standard encryption schemes include DES [2] and IDEA [3]; signature schemes include MD5 [4] and SHA [5]. To achieve acceptable performance, any encryption or signature operations on heavily used communication paths require a shared secret key.

Distribution of these shared keys is central to the correct functioning of a VPN. One must address key expiration, and periodically switch encryption keys. Also, the membership of the VPN is important so that only legitimate users receive up-to-date keys. Key revocation must be possible — when a user ceases to be trusted, his VPN access must be revoked. For example, if a computer is stolen from a laboratory, access from it to the CHIN should subsequently be denied.

The big advantage of a VPN is transparency. Applications are typically oblivious to the fact that encryption and signing are performed under the covers. This permits use of unmodified legacy software, and also follows good software engineering practices by separating security functionality from other aspects of the application.

3 Related Work

The notion of virtual networks was proposed by Bellovin in [23]. Secure encapsulating drivers were introduced by Ioaniddis and Blaze [24]. They use an encapsulating driver in the kernel to create secure point-to-point connections between machines. Our work differs from theirs in many ways: smooth re-keying, the notion of grouping machines into DVPNs, allowing a machine to exist in several DVPNs simultaneously, and the notion of inter-DVPN policies.

Our work exhibits some resemblance to an application level solution proposed by Li Gong [21]. Gong creates a security context in which applications can communicate securely. All multicast communication is encrypted and signed using a symmetric key. His work, although using a similar notion of grouping, is limited to applications implemented over an API provided by the *Enclaves* software. Furthermore, the security context is managed by a member acting as leader. Should it fail, the shared context is lost and the group cannot recover.

The SURF [18] firewall is an attempt to implement an "Academic Firewall". SURF is a VPN substantially more permissive and flexible then commercial VPNs, constructing a security perimeter around groups of machines in a fairly rigorous manner. However, it does not address firewall fault tolerance nor dynamicity concerns.

We are using Ensemble, a secure group communication system. The Clique [22] project is also conducting research on secure group communication. However, their approach focuses on fair key agreement protocols, and the security of existing Internet protocols.

4 Model

A VPN is said to exist between a set of machines if they can communicate without outside interference, or eavesdropping, and if they possess network services indistinguishable from private, dedicated network services. We distinguish between machines that are authorized to join a DVPN $\mathcal{V}$ which we will refer to as the honest machines (in context of $\mathcal{V}$), and the remainder, potentially dishonest machines. We assume no Byzantine faults; Thus, all machines in a DVPN trust each other and mistrust all machines outside the DVPN. Once a machine is allowed into a DVPN, it is trusted until it leaves. Furthermore, a machine is trusted not to leak information received within the confines of a DVPN to the outside.

Our goal is to protect all message content sent between machines in a DVPN. We do not attempt to eliminate covert channels (such as timing channels), nor do we provide protection against retransmission or denial of service attacks.

Security is provided on a machine granularity (since most off the shelf operating systems could readily be compromised by a knowledgeable user). We are thus vulnerable to any security problems inherent to the OS.

We consider only IP communication, and work with an existing IP stack, hence, we are exposed to attacks at the level of the stack itself (although our solution could be strengthened in this respect). For example, a "poison pill" attack might be able to crash a node on which our DVPN is runs, and a flooding attack could saturate its interfaces. We assume the existence of a public key infrastructure and authentication mechanism that allows any machine to authenticate any other machine with complete certainty. This infrastructure, comprising of highly secure authentication and authorization servers, may also be used to pass authentic and confidential information between any pair of machines.

We assume that the adversary may listen to, modify, and retransmit any packet on the wire, it also has control over dishonest machines. However, the adversary does not gain access to honest machines. The adversary may try to crack (discover) encryption keys via search of the key space. With today's technology a 56-bit DES key can be assumed intact for several hours under a very aggressive attack. A 128-bit IDEA key is virtually unbreakable at the time of this writing. The security concern is therefore key disclosure by some leakage.

Each machine p is assumed to have an IP address p_r in the regular IP address space. Virtual addresses for p are denoted by p_v for DVPN $\mathcal{V}$, p'_v for $\mathcal{V}$', etc.

5 Our Solution

Our solution consists of four software modules: a loadable device driver and a packet filter [2], which reside in the kernel, two modified library procedures [3] (**gethostname** and **gethostbyname**, that

[2]Work in progress.
[3]Work in progress

map machine names to IP addresses) and a management **mngr** server that runs with super-user permissions on the host machine. At present, we instantiate all components on every machine where the DVPN is used. We begin with a brief overview of the role of these components, then discuss some of them in greater detail. A schematic diagram is shown in Figure 3; The IP protocol stack is shown sliced according to the ISO layer model. The shaded areas are those which we have added. In the link layer we show the regular Ethernet driver and the DVPN driver. The network layer is composed of the IP layer and the IP filter; with a modified routing table. The transport and session layers have not been modified. Above these are the standard library functions of which we have modified **gethostname** and **gethostbyname**. At the top are the applications, these remain unchanged, except for the addition of the **mngr** server.

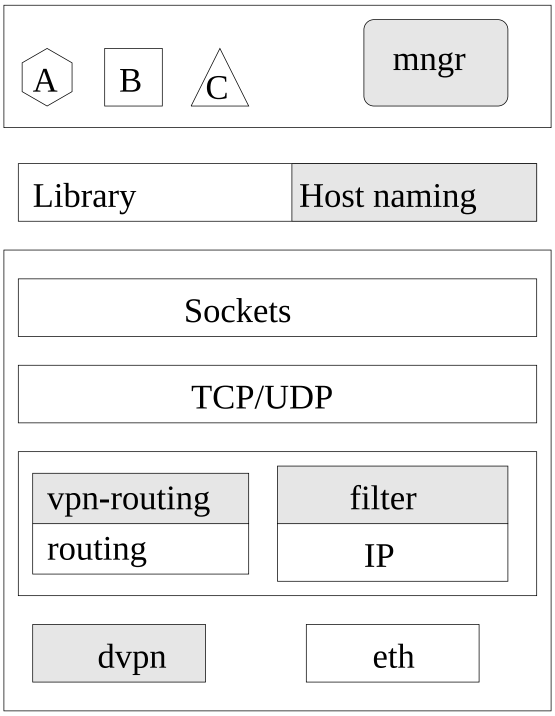

Figure 3: Our security architecture, the shaded areas are those which we have added.

Our DVPNs use a global service which we call the Available Address Service (AAS). This service is provided by a set of servers responsible for allocating virtual IP addresses within the global pool of available virtual IP addresses (see Section 8).

The life of a process can be summarized as follows. The process is first created on some machine, inheriting environmental attributes from the process that launched it. The process may look up the textual name of the machine on which it is running (by calling **gethostname**), and then map this name to its own IP address by calling **gethostbyname**. The process can also obtain the names of other processes from its runtime environment, the network information service, or (later) through interaction with other processes and services. These, too, it maps to IP addresses using **gethostbyname**. The process then communicates using IP services that operate using IP addresses, not plain-text names.

Our DVPN approach starts by arranging that each process is created within [4] some DVPN. More precisely, the login program on the machine is modified to join a DVPN at the time the user logs in. If the login is successful, a DVPN identifier is stored into the process context of the user's command shell, and will be inherited by its child processes, which will pass it to their own children, and so forth.

We virtualize IP addresses and machine names. A machine may have multiple names — its regular DNS name and one for each DVPN to which it belongs. An IP address is associated with each name, a regular IP address for the real name and virtual addresses for the virtual names. For example: host `swift.cs.cornell.edu`, a member of DVPNs sec_1 and sec_2 will have names: `swift.cs.cornell.edu`, `sec1.swift.cs.cornell.edu`, `sec2.swift.cs.cornell.edu`. And corresponding IP addresses: 128.84.248.55, 200.200.200.1, and 200.200.250.3. The **gethostname** function has been modified so that if a process within DVPN $\mathcal{V}$ (identifiable by the DVPN id associated with p in the process table) requests its name, then it gets the name associated with $\mathcal{V}$. For example, a process in DVPN sec_2 would get `sec2.swift.cs.cornell.edu`. The **gethostbyname** function has been extended to handle DNS names in the virtual networks, such names are not recognizable by the regular resolver.

Our packet filter intervenes on all IP communication pathways (functionality of this sort is standard in kernel-level packet filters). For packets sent by a user-level application in DVPN $\mathcal{V}$, the filter checks the source IP address and makes sure that it is correct. It must be the machine's IP address in $\mathcal{V}$. The filter next examines the source and destination IP addresses as a pair. Messages sent within a single DVPN are passed to the IP routing layer. Currently, messages cannot be sent across DVPNs; thus, a process is only aware of other processes within its DVPN. This will be relaxed in the future: we plan to introduce mechanisms for controlled communication between a DVPN and the outside world, or between different DVPNs (see Section 11).

When we install a machine in a DVPN, we modify its IP routing table so that our DVPN driver will be shown as the interface to use for all outgoing/incoming IP packets in the DVPNs.

The role of our **mngr** server is to manage a distributed database of information associated with each DVPN that the machine has joined. It is also responsible for acquiring an IP address in each DVPN it joins from the AAS. As membership of a DVPN changes, the routing table must be adjusted accordingly and the new membership data exported for use by **gethostbyname**. Additionally, the **mngr** runs a protocol by which symmetric keys are generated, and periodically refreshed, for each DVPN. As keys are refreshed, the server installs them in our encapsulating driver by means of a system call.

The sections that follow give additional details for some of these components.

6 The driver

Our driver, D, is placed below the IP networking layer in the IP stack. We describe its functioning for a single DVPN $\mathcal{V}$. Each packet that travels through D is encapsulated within another IP packet. The new packet is then handed back to the IP layer which forwards it to its (possibly) new destination. The destination receives the packet, passes it to the IP layer which decapsulates it and hands to D. The driver then strips its own header and passes the inner packet back to the IP layer. It proceeds to deliver it to the application. Our driver never buffers packets. Packet size

[4]Work in Progress.

is limited according to the link technology used, for example, Ethernet allows no more than 1536 bytes per packet. The driver enlarges IP packets by attaching a new header, thus, it publishes the maximal packet size it can handle without fragmentation. The IP layer automatically fragments packets that are over that limit before handing them to D.

We configure the machine to use our driver by setting up the IP routing table so that $\mathcal{V}$ is accessible through D only. Any IP packet sent to a virtual address will be forwarded through the driver. The driver uses a translation table by which it reroutes packets. A packet from p_v to q_v is sent with a header $p_r \rightarrow q_r$. The encapsulated packet specifies (in the protocol field of the IP header) a new protocol type notifying the receiver that this packet uses a special protocol. The receiver's IP networking layer will demultiplex the IP packet to D, as opposed to the UDP/TCP layers, or the ICMP daemon. Addresses in $\mathcal{V}$ are not visible to the outside for two reasons: we do not publish them through regular IP services, and because they are accessible only via our special protocol and driver.

The packet is encrypted using IDEA, protecting its content from disclosure. Encryption does not guarantee authenticity, thus we also sign the packet with keyed-MD5. This flavor of MD5 starts the MD5 state with a secret key known only to DVPN members. Thus, unauthorized members cannot tamper with the packet signature. The same key is used for encryption and signature. In order for the receiving driver to know which key was used by the sender, all keys are uniquely numbered and the key number used to sign a packet is added to its header.

The header size used is 40 bytes long: 20 giving a normal IP address, and 20 split between 16 bytes for the MD5 hash, and 4 bytes giving the key number.

6.1 Key security

If an encryption key is used frequently over a long period of time, the potential of the key to be cracked or disclosed rises. Therefore, our DVPN switches keys rather frequently. However, this creates a technical challenge, because the network is asynchronous and we do not wish to stop communication while rekeying the DVPN. Suppose, for example, that machines p and q are communicating using key x and it becomes necessary to change to key y. A situation could easily arise in which p possesses the new key and switches to it, but q has not yet received the new key and expects packets to be sealed with x. Machine q would now drop all packets from p, potentially disrupting communication for an extended period of time. Alternatively, p could delay use of key y until it is sure that q has received and switched to y, but now communication from p to q would need to be delayed during the period of uncertainty. For modern protocols, which are typically optimized to assume a low rate of IP packet loss, such disruptions can have serious performance impact. Even with a near simultaneous rekeying, a related issue would arise because packets may be arbitrarily delayed in the network. The TCP protocol, for example, assumes a delay bound of 2 minutes and long delays are often observed in wide-area settings. Ideally, we would prefer that the DVPN continue to accept such messages for a brief period. To address both issues, our driver keeps a backlog of n keys. We can choose n to be as large or as small as we wish. In order to switch keys smoothly the backlog must include at least the last key. When a key switch from x to y occurs, a driver starting to seal messages using y still keeps x in order to unseal messages sent by other drivers still using x. Currently we set n to 2.

Please see the appendix 14.1 for more details about the exact format of the tables used by our driver

7 The mngr

The **mngr** is the management daemon running on each machine in the DVPN. It is an application written using our Group Communication System (GCS) Ensemble. This section first describes Ensemble and then describes the role and inner workings of the **mngr**.

7.1 Group Communication

Using a Group Communication System (GCS) to run our **mngrs** greatly simplifies our servers as well as allowing them to receive accurate membership information, central to DVPN management.

A group communication system implements the abstraction of process groups, handling group membership changes and supporting reliable multicast [1]. Communication is asynchronous and packets may be lost or reordered on the network. Communication links may fail and machines may crash or become disconnected. A GCS's role is to overcome these sorts of complex failure scenarios, providing reliable group send and multicast capabilities as well as membership notifications when processes join, depart, or fail. The GCS assists group members in managing the state of the group; this is done using a state-machine [16] style of updates to modify the data, and by employing a state transfer protocol when membership changes occur. When components merge, each component is given the opportunity to prepare a representation of its state, and the set of component states is then presented to the members of the new group, which combine them into the new state. Merging is application-specific; the next section discusses our use of it here. A secure GCS includes mechanisms by which group abstractions can be protected against some classes of attack, for example by ensuring that only trusted processes may join a process group.

The GCS we are using, Ensemble [15, 25], has been developed by our group at Cornell University. It is a successor to the Isis and Horus systems, incorporating ideas from Transis as well [26]. It has a modular flexible structure allowing the user to dynamically choose the set of communication protocols desired.

Generally speaking, membership changes in Ensemble are constrained such that they are either partitions or merges. A partition occurs when a group splits into several disjoint subsets due to network disconnections. A partition generalizes the notion of machine failure. If machine p in group G crashes then we say that the group has split into two components: $G \setminus \{p\}$ and $\{p\}$. A merge occurs when disjoint group components merge. A merge generalizes the notion of *join*; if machine p wishes to join group G it first creates a singleton component of G consisting of itself. It then merges with the pre-existing component of G.

7.1.1 Secure Groups

A *secure process group* allows only authenticated and trusted members to join it. All communication between group members is protected from tampering, eavesdropping, and retransmission.

In Ensemble, we assume that the trust relation is an equivalence relation. We do not handle Byzantine faults but assume that all group members, once admitted, are honest. The Ensemble security architecture is described in more detail elsewhere [17], but we summarize it here. Ensemble supports a number of encryption, signing, and authentication mechanisms and new ones are easy to add. For encryption, it supports DES and IDEA. For signing, it provides MD5. For authentication, it supports PGP and Kerberos. For the work described here, we used PGP for authentication, IDEA and MD5 to secure intra-group communication.

At any time, an Ensemble group may be divided into a number of components. This partitioning may have been caused by earlier network and process failures. Each component uses a symmetric key for signing and encrypts all messages sent between its members, furthermore, all messages carry nonces or counters to prevent retransmission.

When a component partitions into several sub-components, sub-components continue using the old group key. Over time such components will use different keys preventing them from "hearing" each other and subsequentally merging. We use the *Exchange* protocol (see below) to securely establish a common group key between components using different keys.

The key exchange protocol is run by an elected member (called the coordinator) of each component, and works by establishing common keys. Each Coordinator periodically broadcasts its principal name[5] to a well-known multicast address. When coordinator v receives such a multicast from coordinator u, v first checks an application-defined policy to find if u is trusted. If so, then v sends u its name along with a a cryptographic ticket containing the symmetric key for v's component. This ticket is prepared using PGP, it can only be unsealed by u and it authenticates the ticket's origin as v. If u also trusts v, it unseals the ticket and triggers its component to change to using v's symmetric key. There is a danger that u and v will hear each other at the same time and send tickets to each other; to prevent this we break symmetry using endpoint names, if $u > v$ then u sends v a ticket (and vice-versa).

Our security approach has the benefit of using (fast) symmetric keys on the regular message path; and using strongly secure (slow) public keys to disseminate them.

Ensemble also allows changing the group key, we use the Rekey protocol for this purpose. See Section 14.2 in the appendix for more details.

7.2 The mngr server

We can now describe the implementation of the server component of our DVPN architecture. At each machine in the DVPN, a server called **mngr** is started up as part of the initialization procedure. The servers join a secure process group that manages $\mathcal{V}$. This includes supplying the drivers throughout $\mathcal{V}$ with identical translation and key tables. The translation table simply keeps a mapping between a group member its virtual IP address, real IP address, and real DNS name. This mapping is updated every view change. The key table simply keeps the current keys. A specific concern occurs upon a view change. To avoid communication disruptions we'd like to transition smoothly to a single key. We begin by merging all the key tables into a single table. After two minutes we switch to a new key that the (possibly new) leader distributes and erase the others.

The **mngr** also schedules the following events periodically:

Key Distribution: We rekey the DVPN by having the leader select a new (random) key which it securely broadcasts in the **mngr** group. In order to ensure that a key x will not be used until all members obtain it, we rely upon a two-phase-commit protocol for each key update. The leader chooses x and broadcasts it to the group. All the members send a message acknowledging receipt. The leader then broadcasts a *Commit* message informing all members

[5] This is the unique name by which the authentication service knows it.

that they can begin sealing packets using x. If a membership change occurs during this protocol then the key is commited through the view-change/state-transfer protocol.

DVPN Sync: This operation erases all old keys, leaving only the newest key in place. When this process is done, only members in the DVPN know the current key. Thus, no old member can eavesdrop on current communication.

Mngr group rekey: Every few hours we refresh the Ensemble group key (in distinction to the DVPN key). We simply trigger the Ensemble Rekey protocol to perform this.

Normally, the above three operations are performed periodically by the **mngr**. However, the user may also request them at his discretion.

7.3 Key Lifetimes

The security of our solution depends upon a hierarchy of keys and assumptions. The least sensitive keys are also the ones used most heavily, namely the keys employed in the DVPN drivers. These are currently 128-bit symmetric IDEA keys. We rekey at this level once per minute, measured by the leader's clock. If a key is somehow compromised or cracked — the exposure is limited to data transmitted while that key was still in use. More sensitive are the keys used within the **mngr** group and managed using the Ensemble security protocol. These are also 128-bit IDEA keys; if one were broken, the intruder would be able to decrypt the driver-level keys distributed with it. If the current group key is broken the intruder would also be able to break the GCS by forging bogus messages and breaking communication protocols. Since these keys are used much less frequently and in a very restricted context, their lifetime can be longer, currently, several hours. Of course, if a machine leaves the DVPN or appears to fail, but may actually be at risk of compromise, it would be desirable to rekey promptly. Finally, we depend upon the public key infrastructure. Were a private RSA key to be compromised, this would compromise the past history of all groups that the corresponding machine was able to join. Future communication in those groups would be at risk until the next rekey event.

8 Available Address Service (AAS)

We create a global service to administer virtual IP addresses. This service is similar to the DNS in that it binds Internet names to IP addresses. However, it does not answer naming queries. The local **gethostbyname** function receives such information through its local **mngr** server.

We instantiate a server at all LANs in which machines use some DVPN. The AAS servers split amongst themselves the global pool of available IP addresses per DVPN. Each server has authority over its slice of the addresses. Each machine must receive an IP address from its local AAS for each DVPN it wishes to join. The AAS keeps track of the virtual addresses it has given out and the machines which have received them.

Currently, the mapping from text name to virtual address is static, but we intend to make them dynamic in the future. This could be useful in alleviating IP address shortage on virtual networks. Such an approach has already been used in many PC networking products, such as Microsoft's VPN.

9 Performance

We performed several performance tests including continuous runs of the system for several days. The tests were run between two 200MHz PentiumPro machines connected through $100Mbit/sec$ Ethernet, the machines were situated on the same Ethernet segment. To perform the tests we used a *Virtual Network* driver (vn-driver); it is identical to the vpn-driver except that it creates virtual addresses without the added cost of encryption and signature. We also used an MD5 driver (md5-driver) which is the same as the VPN driver except that it does not perform encryption.

First we checked our machine's raw capacity for encryption and signature. The cost per-byte of IDEA encryption is $0.55\mu sec/byte$, for MD5 hashing it is $0.05\mu sec/byte$, see the appendix 14.3 for a detailed analysis.

We measured the latency achievable using our driver (Figure 4) using varying packet sizes in multiplies of 1460 bytes (maximal packet size for our driver). The x-axis shows packet size in bytes, the y-axis shows latency in seconds. We see that the MD5 and vn-driver are fairly close to regular IP communication. On the other hand, the driver that uses IDEA is fairly costly. The difference in performance can be attributed solely to encryption cost. The cost per packet of size 1460 is the sum of encryption (sender) and decryption (receiver) costs:

$$2 * 0.55(\mu sec/byte) * 1460byte = 1606\mu sec$$

The measured difference is asymptotic to this number. We conclude that adding encapsulation and virtual addresses incurs a very low performance overhead. This makes our design suitable for providing message integrity; one can make a complete IP network strongly secure from tampering using our scheme. Since the cost of sending a packet through the VPN driver is the cost of sending it through the regular IP stack, adding encryption, then the cost of using a DVPN is the cost of encryption. Using a high performance CPU or dedicated hardware should alleviate this cost, allowing applications to use DVPNs at nearly no cost.

We went on to check achievable throughput. A TCP connection was opened between two machines and 10 megabytes of data were transmitted through it. We ran this 3 times and averaged. Table 2 shows throughput in $Mbyte/sec$ in four cases: using the regular IP stack, vn-driver, md5-driver, and vpn-driver. We performed this test over a $10Mbit/sec$ Ethernet as well. In the $10Mbit/sec$ case[6], using the vn-driver introduces insignificant cost while the vpn-driver reduced throughput to about 90%. On a $100Mbit/sec$ Ethernet the picture is very different. Using the vn-driver reduces throughput to 97%, the md5-driver to 81% and the vpn-driver to 17%. The bottleneck in this case is the CPU. Adding MD5 and IDEA adds to the CPU overhead and prevents it from processing incoming packets. There is no idle CPU time to use for encryption and signature.

For more performance measurements please see the appendix 14.3.

10 Scalability

The Ensemble infrastructure on which we are building our solution currently scales to approximately 100 machines spread across a local area network. This allows **mngrs** on up to 100 machines to

[6]Our department switched to using $100Mbit/sec$ Ethernet while work was in progress; hence, we do not have complete measurements.

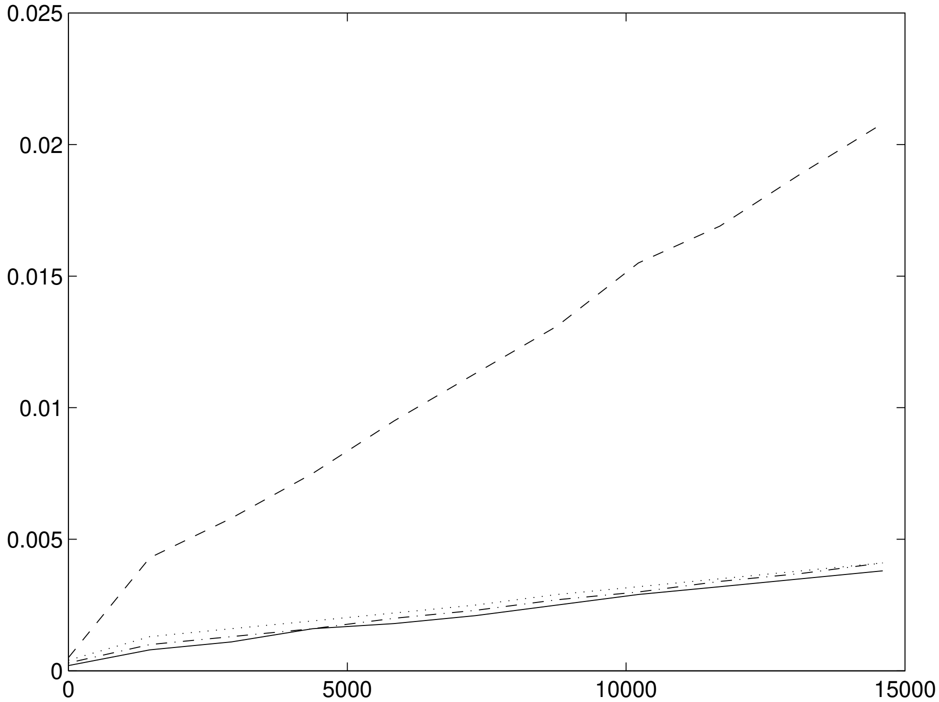

Figure 4: Latency using: no driver, vn-driver, md5-driver, and a vpn-driver. The x-axis shows packet size in bytes, the y-axis shows latency in milliseconds.

	Maximum	regular	vn-driver	md5-driver	vpn-driver
100Mbit/sec	12	8.823	8.568	7.114	1.486
10Mbit/sec	1.2	0.734	0.731	NA	0.666

Table 2: TCP throughput in megabyte per second for $10Mbit/sec$ and $100Mbit/sec$ Ethernet. The columns show, from left to right, maximum achievable throughput, and then throughput using: no encapsulation, vn-driver, md5-driver, and vpn-driver.

communicate securely and reliably. The question then is whether the *protocols* that we use are scalable to networks of this size.

The DVPN rekey protocol costs two broadcasts and n point-to-point messages. It's latency is 3 rounds of Ensemble communication. This protocol is the fastest and most heavily used. It takes less than 10 milliseconds to complete for 10 machines and can scale well.

In the state transfer protocol we merge the key and translation tables. All members send their state to the leader which merges and broadcasts the new state. This poses a total communication load of n point-to-point messages and one broadcast. The latency is 2 rounds of Ensemble communication. It is executed when a view change takes place; Ensemble takes less than a second to perform a view change, we add a fairly light overhead.

As the number of machines grows, we may have to make this protocol more scalable, this could be done using a tree structure to send the acknowledgments back to the leader.

We believe that our protocols scale, at least to a 100 machines in a LAN. We do believe that they can scale to much more than that, using a client/server architecture for Ensemble and some work on the data structure used to store the translation table. We will pursue this course in the future.

We had done some preliminary testing with several superimposed DVPNs on a machine. We observed no performance degradation and we believe that our design scales on this axis. Using multiple DVPNs facilitates fine-grained security.

11 Future Work

IPsec includes all the encryption and signature options implemented by our driver. When it becomes available our encryption and signature code will no longer be of use. Instead, the driver will set the appropriate IPsec options for each outgoing packet

We have restricted applications in a DVPN to communicate only with other applications on the same DVPN. We intend to alleviate this restriction using a policy inside the packet filter. For example, a (weak approximation of) multilevel security architecture could be built using a concentric set of DVPNs using policies that allow applications to receive information from all less secure DVPNs, and send information up to any higher security DVPN.

12 Conclusions

We have presented a DVPN architecture which we believe to be significantly more flexible than traditional VPN systems. Unlike more standard VPN approaches, our solution dynamically rekeys the DVPN rapidly (as often as once every minute), and creates a self managing DVPN that can survive network faults and partitions. An initial implementation of our system already demonstrates that the approach is feasible, has low performance overhead, and can be carried out with minimal, rather standard, OS modifications. Looking to the future, we plan to gradually extend the functionality of the solution by supporting overlapping DVPNs with user defined security policies, and scaling our solution to WANs. The software implementing our DVPN will be available through the public domain. The distribution web site is: http://www.cs.cornell.edu/Info/Projects/Ensemble

At the time of writing, the driver, management server, and AAS server have been completed, we are currently working on the IP-filter and naming functions.

13 Acknowledgements

Our work benefited from discussions with Robbert van Renesse, Tal Anker, and Yaron Minsky for which we are very grateful. Zhen Xiao developed the scalable version of the Rekey protocol and Robbert helped debug our driver.

References

[1] K.P. Birman. *Building Secure and Reliable Network Applications*. Manning Publishing Company/Prentice Hall, Jan. 1997. http://www.browsebooks.com/Birman/index.html.

[2] Data encryption standard. National Bureau of Standards, Federal Information Processing Standards Publication 46, Government Printing Office, Washington, D. C., 1977.

[3] X.Lai, J.L.Massey, and S.Murphy. Markov Ciphers and Differential Cryptanalysis Advances in Cryptology – EUROCRYPT '91, *Lecture Notes in Computer Science 547*, Springer-Verlag,1991.

[4] R. Rivest. The MD5 Message Digest Algorithm. RFC 1321, SRI Network Information Center, April 1992.

[5] Secure Hash Standard (SHS). National Institute of Standards and Technology (NIST). FIPS Publication 180, May 1993.

[6] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[7] Digital Signature Standard (DSS). National Institute of Standards and Technology (NIST). FIPS Publication 186, May 1994.

[8] J.G. Steiner, B.C. Neuman, J.I. Schiller. Kerberos An Authentication Service for Open Network Systems. In *Usenix conference proceedings*, Dallas, Texas, February 1988, pages 191-202.

[9] Security Architecture for the Internet Protocol. RFC 1825, August 1995.

[10] IP Authentication using Keyed MD5. RFC 1826, August 1995.

[11] IP Authentication Header. RFC 1828, August 1995.

[12] P.V.McMahon. SESAME V2 Public Key and Authorization Extensions to Kerberos. ISOC Symposium 1994.

[13] W. R. Cheswick and S. M. Bellovin. *Firewalls and Internet Security Repelling the Wily Hacker.* Publisher: Addison-Wesley Professional Computing Series, 1994.

[14] D. Eastlake and C. Kaufman. Domain Name System Security Extensions. Network Working Group, RFC 2065, January 1997.

[15] R. Renesse, K. P. Birman, M. Hayden, A. Vaysburd and D. Karr. Building Adaptive Systems Using Ensemble. Cornell University Technical Report, TR97-1638, July 1997.

[16] F.Schneider. The state machine approach: a tutorial. Technical Report 86-800, Department of Computer Science, Cornell University, December 1986. Revised June 1987.

[17] M. Hayden and O. Rodeh. Ensemble Security. Cornell University Technical Report. Forthcoming in 1998.

[18] M.B. Greenwald, S.K. Singhal, J.R. Stone, D.R. Cheriton Designing an Academic Firewall: Policy, Practice and Experience with SURF. Stanford University, 1995, CS-TR-95-1558

Entry	Real IP
1	127.1.1.113
2	123.4.3.1
3	90.30.12.43

Table 3: Example of a translation table

[19] K. Hamzeh, Ascend Communications, G. S. Pall, Microsoft Corporation, W. Verthein, U.S. Robotics/3Com, J. Taarud, Copper Mountain Networks, W. A. Little, ECI Telematics. Point-to-Point Tunneling Protocol–PPTP. Internet Engineering Task Force (IETF), July 1997.

[20]

[21] L. Gong. Enclaves: Enabling Secure Collaboration over the Internet. *6th Usenix Security Symposium proceedings*, San Jose, California, July 1996, pages 149-159.

[22] M. Steiner, G. Tsudik, M. Waidner Cliques: A New Approach to Group Key Agreement The Information Sciences Institute, The Cliques project, September 3,1997

[23] S. M. Bellovin. Pseudo-Network driver. *Usenix January 1990.*

[24] Ioaniddis and Blaze. swIPe. *4th Usenix UNIX Security Symposium 1993.*

[25] M. Hayden. *The Ensemble System.* Phd Thesis, Department of Computer Science, Cornell University, January 1998.

[26] Y. Amir, and D. Dolev, and S. Kramer, and D. Malki. Transis: A Communication Sub-System for High Availability. *22nd FTCS conference*, July 1992, pages 76-84.

14 Appendix

14.1 Tables

This section describes the key table and translation table kept by the device driver.

The translation table is an array. We currently limit the choice of virtual addresses in $\mathcal{V}$ to a modification of the last byte. This provides an address space of 254 addresses. For example, for network 200.200.200.0 we get the address space between 200.200.200.1 to 200.200.200.254. The translation table is then simply an array of size 256 containing for each array entry k the real IP address for the virtual IP address $V.k$.. If virtual address 200.200.200.1 is at real IP address 127.1.1.113, 200.200.200.2 at 123.4.3.1, and 200.200.200.3 at 93.30.12.43 then the address table will look like Table 3. This makes the conversion from virtual to real addresses, needed at the sending side, very fast.

Keys are created by members of the DVPN. They are numbered uniquely by multiplying the creator's virtual address by the key window size (n) and adding the key serial number. For example, if the creator's virtual id is 128, and the possible serial numbers are 0 and 1 (n being 2), then keys it creates will be numbered $256 = 2 * 128 + 0$ and $257 = 2 * 128 + 1$. In the normal case there is only one key creator in the whole DVPN, the leader, thus the driver need only keep a very small set of keys. However, as detailed in Section 7.1, when a network experiences partitioning failures additional keys can be generated, and will then need to be retained for a period of time after the partition heals and the two DVPN components merge.

Upon receiving a packet the driver computes the key with which it was signed and encrypted. The key number is split using a modulo n operation, into the creator's virtual id and the key's serial number. The key is then retrieved from the key table and the packet is verified and decrypted.

Key serial numbers wrap around with time and key numbers get reused. However, we do not accept packets signed with old keys. For example, assume that D receives a packet signed with old key number 1. The packet is unsuccessfully verified with current key number 1, and dropped.

14.2 Ensemble Rekey Protocol

This section describes the Rekey protocol used in Ensemble to securely switch the group symmetric key.

Ensemble group members trust each other while they share a symmetric key, so it is not necessary to rekey the group each time the membership changes. However, we do rekey an Ensemble group periodically (to keep keys from expiring), and a rekey can also be solicited by the application. For instance, an application may consider failed processes to be an eventual security hole and initiate a rekey soon after a process has been removed from the group.

In Ensemble's Rekey protocol the leader chooses a new (random) symmetric key encrypts it multiple times, once for each group member using that member's public key, and signs it using the leader's private key. The sealed keys are then sent point-to-point to the members. The members acknowledge receipt and the leader, upon receiving an acknowledgment from all the members, performs a view change that takes the new key into effect. We have extended this protocol to scale better, but brevity precludes a detailed discussion.

This protocol suffers from the high cost of using PGP; a seal/unseal operation on our machines costs between 0.25sec, for a 200Mhz PentiumPro, to 1.7sec, on a SparcLX. However, the Rekey protocol is rarely used and it is performed in the background, hence, it does not affect performance.

14.3 Performance

This section contains some more performance measurements supplementing the results described in the Performance section.

To check our machine's raw capacity for encryption and signature we encrypted and signed messages of varying length. In Figure 5 we see the costs of IDEA and MD5. Each test was performed 50000 times and the results were averaged. The y-axis represents cost in microseconds; the x-axis represents message size in bytes. The graph depicts the relation between message size and function cost per byte. As message size grows function costs go down. For messages larger than 100 bytes the costs are a constant $0.55\mu sec/byte$ for IDEA and $0.05\mu sec/byte$ for MD5. This implies that a machine can encrypt a maximum of 1.8 megabyte of data per second. This is a bit more than maximal usage of a $10Mbit/sec$ communication link; but making complete utilization of a $100Mbit/sec$ Ethernet link impossible.

The low per-byte cost for large objects reflects heavy compiler optimization (we used **gcc -O2**) which is less efficient for small messages. The optimizations also causes the strange curve in the IDEA graph for short messages.

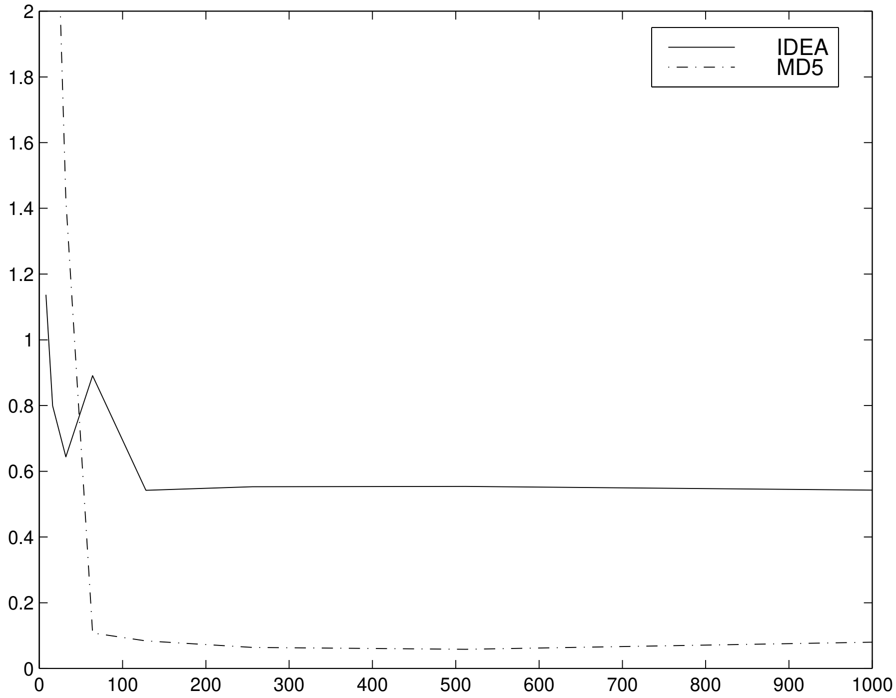

Figure 5: IDEA and MD5 costs in microseconds per byte. The x-axis shows message size, the y-axis shows cost in micro seconds. As messages get longer, cost-per-byte go sharply down.

We also tested the performance of our **mngr** servers using a heterogeneous cluster of machines. We tested the most heavily used protocol — the DVPN-rekey protocol. We used a cluster of different machines since we envision our DVPN operating on many platforms, though currently it runs only on the Linux OS. The following measurement was performed with 11 machines: 3 200Mhz PentiumPro Linux machines, 5 Sparc10/20 Solaris 2.5, 1 SparcLX Solaris 2.5, and 2 SparcLX SunOS 4.1.3..

No attempt was made to optimize the Ensemble protocol stack, since the results obtained using a non-optimal version were sufficient for our purposes. In Figure 6 we see the cost of rekeying the DVPN. Recalling the protocol: the leader broadcasts a new key, the members acknowledge by a point-to-point message and the leader commits the key via another broadcast. We measured the latency at the leader machine, the member machines still need another broadcast to complete the protocol. By multiplying the measured latency by a factor of 2 we can bound their latency. The latency for groups of up to 10 machines is less than 10 milliseconds which we believe to be more than adequate for our purposes.

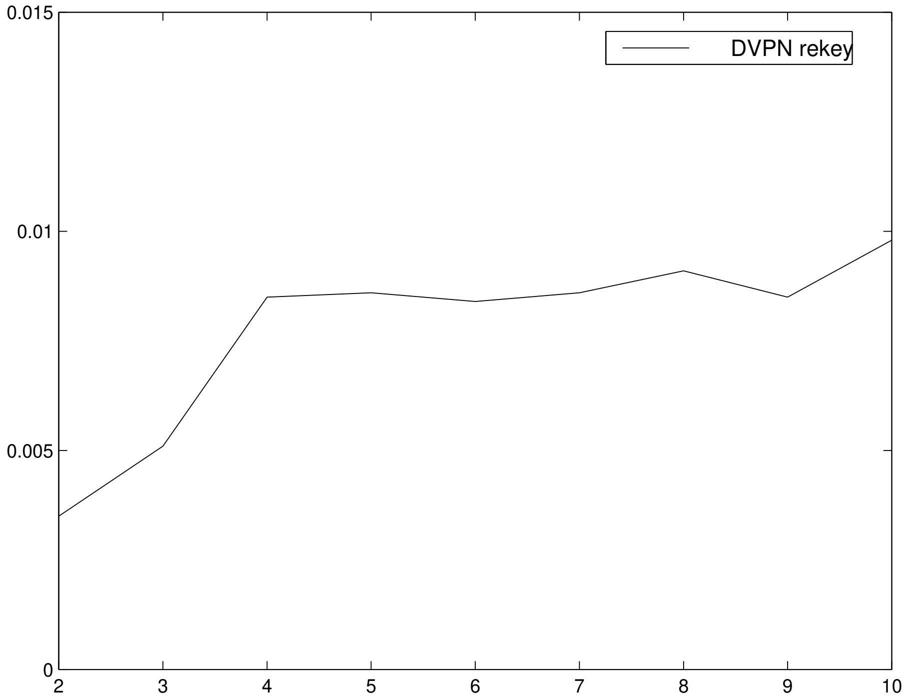

Figure 6: DVPN rekey cost. The x-axis depicts the number of members in the group. The y-axis depicts latency in seconds.