# Virtual Private Networks: An Overview with Performance Evaluation

*Shashank Khanvilkar and Ashfaq Khokhar, University of Illinois at Chicago*

## ABSTRACT

Virtual private networks have gained immense popularity among commercial and defense organizations because of their capability to provide secure connectivity at lower costs. Several commercial and open source VPN products are now available that can be configured to provide VPN services with varying characteristics. This article studies some of the most popular Open-Source Linux-Based VPN solutions (OSLVs) and compares them with respect to network performance (measured in terms of overhead, bandwidth utilization, and latency/jitter), features and functionalities (e.g., algorithm plugins and routing), and operational concerns (defined by security and scalability). Our experiments suggest that there is no single OSLV solution that excels in all considered aspects, and a combination of different VPN products and/or trade-off among desired characteristics may be required to deliver optimal performance. Our experiments also suggest that on an average, OSLVs using UDP-based tunnels have 50 percent lower overhead, 80 percent higher bandwidth utilization, and 40–60 percent lower latency/jitter than those using TCP.

## INTRODUCTION

The Internet is considered the cornerstone of connectivity for organizations to expand their operations and increase revenues. As organizations grow, different methods must be employed to provide secure and efficient connectivity among geographically distributed branch offices, strategic partners, and mobile/telecommuting employees. Although several technologies exist, most recently Internet-based virtual private networks (VPNs) have evolved as the most secure and cost-effective means of achieving these goals. VPNs belong to a family of overlay networks that use IP tunnels to form a virtual network on top of the Internet. These tunnels use cryptographic techniques to provide robust security and privacy. To a remote user, VPN provides all the benefits of a private network, while corporations benefit from the low operational costs, high security, and instant global coverage offered by it [1].

Several commercial VPN products are now widely available, differing mainly in terms of cost and capabilities. In recent years, however, open source VPN solutions have attracted a lot of attention from researchers and developers alike. These are a breed of software solutions that can provide commercial-grade VPN services at commodity prices, using desktop computers running Linux. The availability of source code and open source licensing allows developers to customize and optimize (and sometimes commercialize) these products for specialized hardware to extract maximum performance; for example, SnapGear's [2] LITE+ and SME550 products use a port of FreeS/WAN (an open IPSec implementation) on µClinux (Linux for microcontrollers). Since Linux is used as the underlying operating system, deployment is quick and easy, circumventing the need for system administrators to learn proprietary operating systems. All this coupled with free technical support, available through newsgroups and mailing lists, presents a good alternative for small to medium businesses to operate globally at virtually zero cost. With advancements in Linux powered routers, these solutions, in our opinion, have the potential to carve a niche in the VPN market.

In this article we study and evaluate some of the most popular Open-Source Linux-Based VPN solutions (OSLVs). We present a comparative study, based on network performance, supported features/functionality, and operational concerns, for several popular OSLVs that are freely available over the Internet. From this study we seek to highlight common drawbacks that currently exist in OSLVs and underscore future research questions that need to be addressed. Recently, Pena and Evans [3] have studied different VPNs in terms of throughput and CPU usage in high-/low-speed networks. Although important, we feel that there are several aspects (overhead, security, modularity, etc.) that characterize a VPN solution and hence warrant comparison. Our work is comprehensive in terms of both the number of VPN solutions considered and the number of characteristics compared.

Our study reveals that there is no single OSLV that excels in all considered attributes; selecting an appropriate solution is a complicated procedure, involving many trade-offs depending on the company's policy and intended applications. For


0163-6804/04/$20.00 © 2004 IEEE

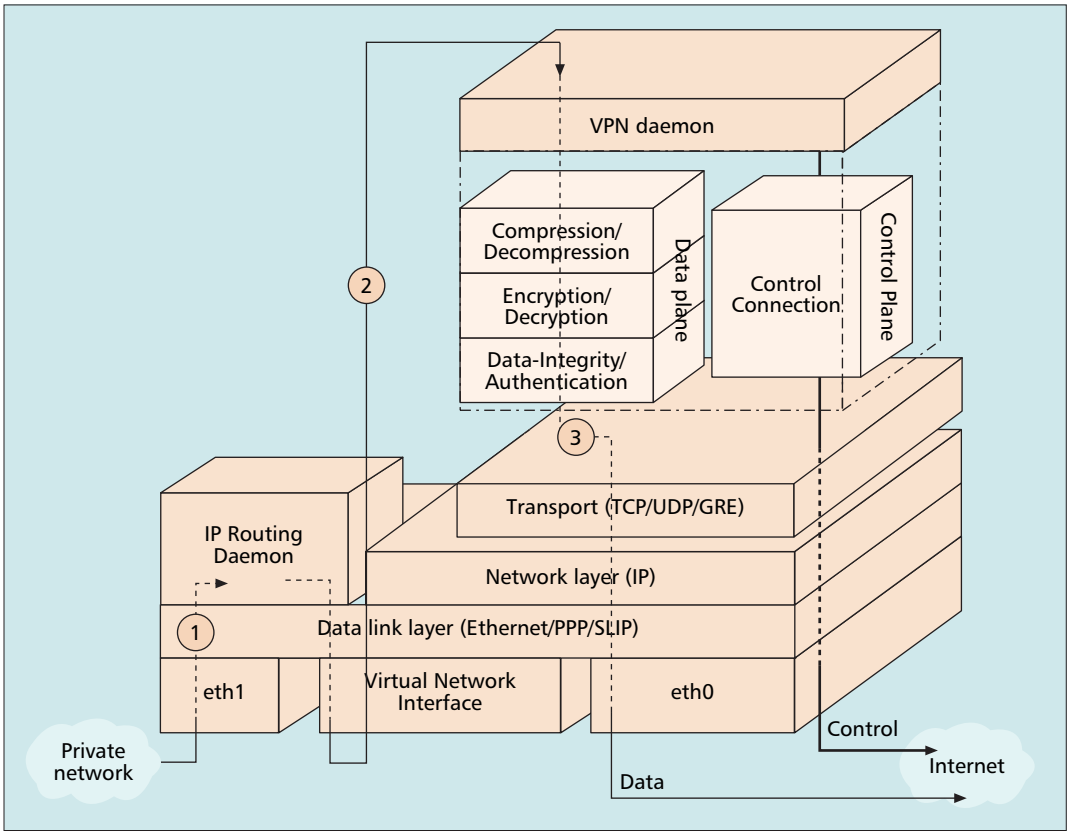


**Figure 1.** *The software architecture of an OSLV router.*

*The Virtual Networking Interface is an abstract device specifically created during VPN initialization to seamlessly exchange packets between the IP routing daemon and the VPN daemon. Any packet routed through the VNI gets automatically delivered to the VPN daemon and vice-versa.*

example, *FreeS/WAN* demonstrates low latency and jitter, making it suitable for real-time applications. However, it introduces high overhead and low bandwidth utilization. Hence, for situations where these factors are important, alternatives like *Cipe* or *PPTP* may be employed. In short, several factors need to be carefully evaluated before an OSLV solution is selected for deployment, and multiple solutions may be made to interwork to extract maximum performance. A condensed version of this work also appears in the proceedings of ICCCN '04.

## The Software Architecture of an OSLV Router

The software architecture of a typical OSLV router is illustrated in Fig. 1. It is a modification of the TCP/IP network protocol stack with two additional components: the *VPN daemon* and the *Virtual Network Interface* (VNI). The VPN daemon is a user- or kernel-level process with a separate control plane for connection maintenance and a data plane for data processing. The control plane is used for peer authentication and generating session keys that are subsequently used to secure tunnels. It also maintains a mapping between IP addresses of peer OSLV routers and private subnets, which is consulted while tunneling a VPN packet. The data plane is like a serial pipeline providing encryption, authentication, and compression services. Control planes generally use TCP for transport, while the data plane may use either TCP or UDP. Later we show that VPNs that use connection-oriented protocols (like TCP) have significantly poorer network performance than those using connectionless protocols (like UDP).

The VNI is an abstract device specifically created during VPN initialization to seamlessly exchange packets between the IP routing daemon and the VPN daemon. Any packet routed through the VNI gets automatically delivered to the VPN daemon and vice versa. Like any real network interface, a data link protocol like PPP or SLIP controls packet delivery over the VNI. The universal tun/tap driver and pseudo-terminals are some popular mechanisms for creating a VNI.

Figure 1 also outlines the data path taken by a packet as it travels from one private network to another, divided into three parts:

1) On arriving at *eth1*, the VPN packet is subsequently handed over to the IP routing daemon, which determines its next-hop router/egress interface by using a longest prefix match on its destination address.
2) Since the destination is a private address, the routing daemon will drop this packet unless the routing table is updated to route it through the VNI, or the routing daemon is modified to recognize a VPN packet and directly hand it to the VPN daemon. Although the second method eliminates an extra trip down the protocol stack, it requires changes to routing protocol, which is much more complicated. Hence, the first method is usually preferred.
3) The VPN daemon treats all packets as data and subjects them to compression and other

*Network performance of an OSLV is measured in terms of overhead, bandwidth utilization, and latency/jitter. It is a well known fact that VPNs have poor network performance, implying high overhead, low bandwidth utilization, and high latency/jitter.*

cryptographic functions. It then determines the public IP/port of the peer OSLV router, wraps it in a new IP header, and sends it as a normal data packet. At the receiver the exact reverse steps take place as the packet is delivered to the correct destination.

## VPN Characteristics

Different VPN characteristics play a significant role in selecting an optimal solution for a given application scenario. In the discussion that follows, we classify these characteristics along the following three dimensions: network performance, supported features and functionalities, and operational concerns.

### Network Performance

Network performance of an OSLV is measured in terms of overhead, bandwidth utilization, and latency/jitter. It is a well-known fact that VPNs have poor network performance, implying high overhead, low bandwidth utilization, and high latency/jitter. Below we identify some root causes of this anomaly and discuss popularly used remedies.

***Overhead*** — The software architecture discussed earlier explains in part the large overhead added to a basic data packet as it travels through several layers of protocol. All OSLVs invariably suffer from this large overhead, 75 percent of which is primarily contributed by the headers/trailers added by the various protocol layers, and the remaining 25 percent by the cryptographic algorithms employed for security. Header/trailer overhead can be reduced through compression. Some OSLVs (like *PPP_over_SSH* and *Stunnel*) use the header compression routines provided by data link layer protocols (like PPP) operating over the VNI, while others (like *Tinc* and *OpenVPN*) provide built-in compression utilities. Compression, however, introduces additional latency and, if blindly used for every packet, regardless of its size or type (compressed or encrypted), may not improve performance [4].

The overhead introduced by the cryptographic algorithms typically cannot be reduced, and is contributed by the specific cipher/medium access control (MAC) algorithm in use. Stream ciphers do not add any overhead, but the more popular block ciphers (like 3DES or blowfish) require the input to be a multiple of 8 or 16 bytes (called *block size*), which adds to the overhead in the form of extra padding bytes. MAC algorithms such as MD5 and SHA1 contribute an additional 16–20 bytes. Other minor contributions are from sequence numbers and timestamps used to defeat replay attacks.

***Bandwidth Utilization*** — This is the maximum bandwidth achieved by a TCP stream and is affected by overhead, latency, and TCP stacking. While overhead reduces the amount of useful bytes transferred, latency affects the bandwidth-delay product for a TCP connection. For long fat virtual pipes (as in the present case), the latency introduced is often so large it is physically impossible to allocate the larger buffers required to support the corresponding increase in TCP window sizes to maintain the bandwidth utilization. The last factor, TCP stacking, refers to using TCP multiple times along the packet data path. This happens if a TCP stream flows over an OSLV that uses TCP-based data channels. Interference between TCP timers at different layers causes this degradation [5]. We found this to be the most profound effect, with OSLVs using TCP performing far worse than those using UDP.

***Latency/Jitter*** — Higher latencies occur due to the elongated packet data path (as apparent from Fig. 1), the compute-intensive nature of cryptographic functions, and multiple copying of data packets between kernel and user space. A good method to shorten data paths is to resort to layer 2 switching, as in multiprotocol label switching (MPLS) or asynchronous transfer mode (ATM), while higher computational time can be reduced by using faster hardware [6] or better cryptographic algorithms. Finally, multiple copying of data packets occurs because OSLVs exist as daemons in the user space and can be eliminated by integrating them into the operating system (OS) kernel. This, however, reduces operating flexibility, as most users are not kernel experts. Similarly increase in jitter depends on the OS scheduling scheme and can only be reduced by increasing its process priority.

### Supported Features/Functionality

OSLVs are expected to support certain important features to enhance performance. Here, the importance of supporting two such features is discussed, while the results section presents how well equipped current OSLVs are to support such features.

***Code Modularity*** — This refers to the flexibility of an OSLV solution when it comes to using algorithms of one's choice. Thus, code modularity can be defined as the OSLV's support for algorithm plugins, which can range from simple plugins for cryptos to more complex ones for routing or security. Algorithm plugins are an excellent way to allow quick and easy integration of newer algorithms into the OSLV solution and gives its users an option to create or buy independently developed plugins for ciphers, digests, or compression. OSLV solutions designed with plugins can survive potential security threats, thus extending their shelf life, and also allow companies to use their own proprietary code.

***Routing*** — Routing needs to be set once the tunnels are established and the topology is set. The routing table in every OSLV router node needs to be updated with information on reaching its peer nodes. This can be done either manually or automatically. Routes can be manually updated using a command line utility like *route* (available on most UNIX systems). However, this presents a cumbersome and often error-prone exercise when the number of nodes is large, and often results in a full mesh topology. Automatic routing employs an independent routing daemon to exchange routing information and configure routing tables. For single-domain VPNs, where the private address space is unique, a separate routing program can be used to update private routing information. This

approach allows arbitrary topologies to be created and proprietary algorithms to be used. For multidomain VPNs, where the address space is not unique, routing becomes a complicated issue. This is currently a hot area of research, and one possible solution is to have separate routing instances and routing tables (called virtual routers) for every VPN.

### Operational Concerns

Selecting an OSLV solution for deployment requires good background information on many operational features like security and scalability, which ultimately decide the usefulness of the VPN solution. These are qualitative parameters that are difficult to define and compare. In this section we present a number of properties for each of these aspects to get a better idea about them.

***Security*** — Although the security offered by an OSLV solution is an important characteristic, it is highly relative and difficult to define in absolute terms. Until the recent past, secrecy and obfuscation was mainly used to provide security, which was manifested in large-scale use of proprietary non-standard algorithms (e.g., MPPE used in *PPTP*) where source code/algorithm is kept a secret. However, this notion has long been abandoned because many such algorithms have already been broken [7]. The new and widely accepted norm is open standardized protocols where the algorithms and their implementations are widely published over the Internet, which allows extreme scrutiny on the strength of the algorithms by multitudes of experts. At this time there are no litmus tests to determine the relative security offered by different protocols, which makes it difficult if not impossible to gauge it. Hence, any preconceived comments on this topic would be inconclusive. However, we have rated standard protocols like SSH, SSL/TLS, and IPSec as more secure than nonstandard protocols. Determining the relative strength of the security offered by OSLVs belonging to the same group is out of the scope of this article.

***Scalability*** — Clercq and Paridaens [8] have analyzed the scalability of provider provisioned VPNs in terms of *memory consumption*, *processing power*, and *configuration and management load*. We use similar lines of reasoning to address scalability concerns in this article. Memory consumption and processing power mainly depend on the number of established tunnels. Every tunnel requires certain state (session keys, certificates, routing information, etc.) to be maintained at each end. The maximum size of this state determines the memory consumption with $N$ tunnels consuming $N$ times the available memory. Processing power, on the other hand, is affected by the compute-intensive nature of cryptographic functions. With higher numbers of tunnels, the slice of CPU time allocated for every tunnel is reduced by the same fraction. Lastly, configuration and management load can be quantified by the total efforts required for editing configuration files, updating routing information, and distributing security keys when new tunnels are added or deleted.

Memory consumption and processing power limit the scalability of only a single OSLV node; this problem can be partially eliminated by using powerful desktops with larger main memory or higher processing power. Since OSLVs are mainly targeted for small-scale businesses (typically consisting of 10–50 sites), these factors do not play a significant role and have been neglected in this article. We only concentrate on the last factor, which in our opinion affects the scalability of the entire system.

*Selecting an OSLV solution for deployment requires good background information on many operational features such as security and scalability, which ultimately decide the usefulness of the VPN solution. These are qualitative parameters that are difficult to define and compare.*

## The Experimental Testbed

The testbed used in our experiments consists of two OSLV routers (OSLV-A and -B) and creates a secure tunnel between two private networks (PN-1, PN-2) located in the same building. OSLV-A contains an Intel Pentium 4 processor at 2.0 GHz, with 512 Mbytes of RAM and running RedHat Linux 9.0. OSVL-B is a lower-end desktop using a 400 MHz Pentium II processor, with 128 Mbytes of RAM and running RedHat Linux 8.0. PC-1 (on PN-1) and PC-2 (on PN-2) are Linux desktops that are used to conduct network performance experiments. The testbed is isolated (no external congestion) and all links use 100 Mb/s Fast Ethernet.

The overhead was measured by first analyzing the OSLV solution, and later verifying it, by capturing live UDP packets of known length. These packets were generated using a simple UDP generator program and captured using a packet snooping utility like *ethereal*. Bandwidth and jitter were measured using *iperf*, while *ping* was used to measure latency. Each experiment was repeated 10 times in both directions, and only the average has been reported. Since there was no one common cipher/MAC that could be used with all OSLVs, experiments were performed by using one of {Blowfish, 3DES or none} for ciphers and {SHA1, MD5 or none} for MAC. Compression was disabled at all levels.

We have grouped the selected OSLV solutions based on the security protocol employed, since VPNs using the same security protocol can be expected to deliver almost similar performances. Accordingly, we have VPNs based on SSH, SSL/TLS, IPSec and other proprietary protocols, as shown in Fig. 2. All OSLVs in the top half of the figure use TCP data channels, while those at the bottom use UDP.

SSH, standardized by the *secsh* working group of the IETF, provides support for secure logins and secure file transfers. *PPP_over_SSH* is the only VPN solution that uses SSH. Both PPP and SSH are mature protocols in widespread use, and a simple VPN can be set up using a single command [9].

SSL/TLS, standardized by the *tls* working group, was initially developed by Netscape to provide secure Web transactions (versions 1 and 2), but over the past few years this protocol has matured into a standard solution for secure communications (SSLv3/TLSv1). *Stunnel*, *AmritaVPN*, and *LinVPN* are the OSLVs that use SSL/TLS, with *Stunnel* and *AmritaVPN* using SSLv3/TLSv1 and *LinVPN* using SSLv2.

IP Secure (IPSec) extends security at the network level. However, IPSec also supports a tunneling mechanism that can be utilized to provide

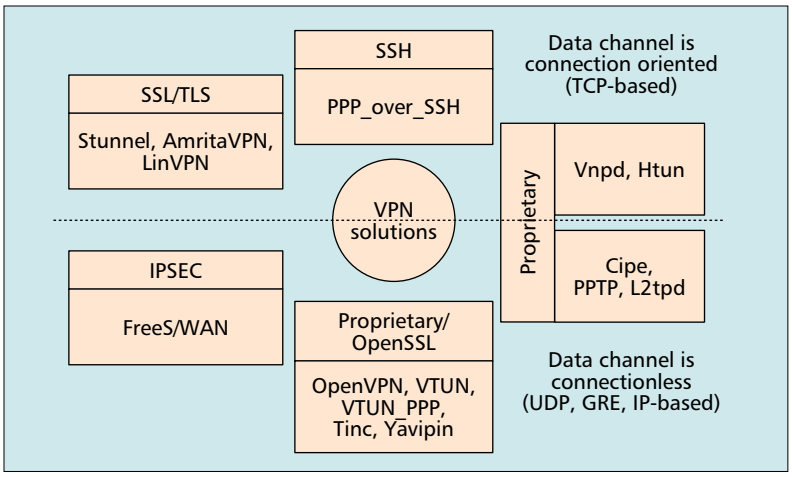


**Figure 2.** *VPN grouping based on security protocol.*

VPN services. *FreeS/WAN* is a Linux-based implementation of the IPSec protocol. It consists of two components; KLIPS and PLUTO. The former offers cryptographic services to IP packets and is built as a kernel module, while the latter is an application-level program used for negotiating session keys subsequently used by KLIPS.

The remaining OSLVs use their own proprietary protocols to provide security. This group has been further divided into those that use standard cryptographic functions exported by OpenSSL (Proprietary/OpenSSL) and those that use their own proprietary implementations (Proprietary). *OpenVPN*, *VTUN*, *Tinc*, and *Yavipin* belong to the first group, while the second group contains *Cipe*, *PPTP*, *Vpnd*, *Htun*, and *L2tpd*. Since *VTUN* can be used with two different VNIs (tun/tap driver and pseudo-terminals), we consider both these organizations as separate OSLVs, with the first referred to as *VTUN* and the second as *VTUN_PPP*. Table 1 provides additional details on each OSLV including links to step-by-step recipes for setting up a VPN. The last column quantifies our experience with setup complexity (a lower number of asterisks imply simplified setup) evaluated against common criteria involving installation, configuration, and support.

## Performance Evaluation Results

In this section we present the experimental results for the selected OSLVs and discuss their performances based on metrics described earlier.

### Network Performance

Figure 3a plots the overhead performance normalized toward 80 bytes/packet, which was also the least overhead added by any OSLV solution. Among them, we found most OSLVs in the proprietary security group (except *Vpnd* and *Htun*) to add relatively lesser overhead than others, making them relatively suitable for applications that generate smaller packet sizes (e.g., telnet and remote login). Overall, OSLVs using UDP-based data channels (GRP_UDP) add 50 percent less overhead than those using TCP-based data channels (GRP_TCP). Smaller differences among different members of the same group are mainly due to different packet formats and random padding appended to the packets. Henceforth, *Htun* has been left out of all performance calculations since it demonstrated network performance that was much lower than the average case. This is partly because of the two-tier nature of its communication protocol.

Average bandwidth utilization is illustrated in Fig. 3b. The numbers on top of every bar indicate the 95 percent confidence interval. Again, all OSLVs were found to perform poorly, with the best one (*Cipe*) utilizing less than 50 percent of the bandwidth. Here, calculating the cumulative average, we find GRP_UDP to be at least 80 percent better than GRP_TCP.

Finally normalized latency and jitter are illustrated in Fig. 3c and 3d, respectively. The normalization is performed with respect to the total latencies (and jitter) observed across each hop when no VPN is in use. *FreeS/WAN* is observed to have the lowest latency and jitter characteristics, making it suitable for real-time applications like voice and video. We believe lower latency/jitter in *FreeS/WAN* occurs because it is integrated in the Linux kernel. Here, GRP_UDP shows 40–60 percent improvement over GRP_TCP.

From the above results it is difficult to judge which OSLV has best overall performance. However, a simple heuristic can be applied to select an optimum solution. For example, by ranking the OSLVs for each attribute (ranking taken from Fig. 3) and assigning weights proportional to the importance of each attribute, one can determine the weighted average rank. Thus, when all attributes are equally important, *VTUN*, *PPTP*, and *FreeS/WAN* (average performance is almost similar) are the best. By varying the weights, the optimum OSLV for any environment can easily be determined. However, this method requires that the network performance for each OSLV be determined in advance.

### Supported Features/Functionality

Earlier we discussed different features and functions an OSLV should provide. Here, we evaluate them based on the features that are currently supported. If a particular feature is absent, we also comment on how simple (or complex) it is to integrate it in the actual OSLV code.

***Code Modularity*** — None of the OSLV solutions strongly support algorithm plugins as defined earlier. Hence, we relax the definition and evaluate the difficulty level with which newer crypto algorithms can be used with an OSLV solution. While considering this aspect, we had to evaluate the general coding style to determine the crypto functions used internally. Accordingly, we have divided the difficulty levels into low, medium, and high as detailed below (refer to Table 2 for actual evaluation).

**Low** — This is the case when the OSLV uses standard third party libraries, like OpenSSL, to implement their security protocols. OpenSSL has a generic application programming interface, called, EVP API, that provides a standard access for all its ciphers. Thus, whenever newer algorithms are invented and integrated into OpenSSL, they can be immediately used without recompiling the OSLV code.

| | Version | Internet standard | Available compression algorithms | Available encryption/ MAC algorithms | Official Web-site | Simple recipes to set up a VPN | Setup complexity |
|---|---|---|---|---|---|---|---|
| PPP_over_SSH | — | Yes | ppp-ccp[1], gzip | All available with SSH, viz. bf-cbc/md5 | http://www.buildinglinuxvpns.net/sourcecode/ | $/ssh-vpn.html | ***** |
| Stunnel | 4.04 | No | ppp-ccp | All available with OpenSSL, viz. 3Des/Sha1 | http://www.stunnel.org/ | $/ssl-vpn.html | *** |
| AmritaVPN | 0.96 | No | None | Cannot specify. Defaults to some value. | http://aitf.amrita.edu/ | $/amvpn.html | *** |
| LinVPN | 2.6-pre1 | No | ppp-ccp | Cannot specify. Defaults to some value. | http://mrtg.planetmirror.com/pub/linvpn/ | $/linvpn.html | ******* |
| Vpnd | 1.1 | No | Available but buggy | bf-cbc / (md5\|sha1\|ripemd160) | http://sunsite.dk/vpnd/ | $/vpnd.html | **** |
| Htun | 0.9.5 | No | None | None | http://htun.runslinux.net/ | $/htund.html | ***** |
| Cipe | 1.5.4 | No | None | (bf-cbc \| Idea)/- | http://sites.inka.de/sites/bigred/devel/cipe.html | $/cipe.html | ** |
| PPTP | 1.2.0 | Yes | ppp-ccp | MPPE | http://pptpclient.sourceforge.net/ | $/pptpd.html | **** |
| L2tpd | 0.69 | Yes | ppp-ccp | None | http://www.l2tpd.org/ | $/l2tpd.html | *** |
| OpenVPN | 1.4.2 | No | Lzo | All available with OpenSSL, viz. bf-cbc/md5 | http://openvpn.sourceforge.net/ | $/openvpn.html | ** |
| VTUN | 2.6 | No | zlib, lzo | Bf-cbc/md5 | http://vtun.sourceforge.net/ | $/vtund.html | *** |
| VTUN_PPP | 2.6 | No | ppp-ccp, zlib, lzo | Bf-cbc/md5 | http://vtun.sourceforge.net/ | $/vtund-ppp.html | **** |
| Tinc | Tinc-CABAL | No | zlib, lzo | All available with OpenSSL, viz. bf-cbc/md5 | http://tinc.nl.linux.org/ | $/tinc.html | *** |
| Yavipin | 0.9.5 | No | Zlib | (bf-cbc \| des-cbc)/md5 | http://yavipin.sourceforge.net/ | $/yavipind.html | ***** |
| FreeS/Wan | 2.01 | Yes | Available | 3des-cbc/- | http://www.freeswan.org/ | $/ipsec.html | ****** |

([1]CCP: Compression control protocol, http://multimedia.ece.uic.edu/volans)

**Table 1.** *Open source Linux-based VPN solutions.*

**Medium** — The difficulty level is considered medium when new algorithms need to be manually inserted in the OSLV code. For example, *Cipe* and *Vpnd* use their own proprietary crypto implementations. Hence, newer algorithms can be used only if they are integrated with the code. This category also includes those OSLVs that directly use OpenSSL crypto APIs instead of the more generic EVP API.

**High** — This means newer algorithms can be used only if substantial changes are made to the source. This occurs if the OSLV did not originally support any crypto methods. Thus, to use newer algorithms one would have to start from scratch. The exception here is FreeS/WAN, included here because the source code seemed too large and complicated to locate where it needs to be modified.

***Routing*** — Table 3 presents our evaluation of the routing mechanisms supported by the current OSLVs, again quantified into:

- ***Manual***: Direct routes must be established manually, using either the *route* command or built-in shell scripts
- ***Automatic (for single-domain VPNs)***: The routing daemon is part of the OSLV solution to handle single-domain VPNs. *Tinc* is the only solution that employs this type of routing, although it still creates a full mesh network. However, here the administrator has to configure only a single site when a new site is added, which makes the solution extremely scalable
- ***Automatic (for multidomain VPNs)***: This employs virtual routing that creates a separate routing instance for every VPN domain. Unfortunately, at this time none of the OSLVs support this option.

## Operational Concerns

***Security*** — As discussed earlier, security of an OSLV cannot be measured. However, many situations can be envisioned where strong security is not a requirement, and people may just settle for weaker protocols. Accordingly, we have rated our confidence in security as:

- ***Open standards***: Includes all OSLV solutions that use standardized security protocols such as SSH, SSL/TLS, and IPSec. We accept

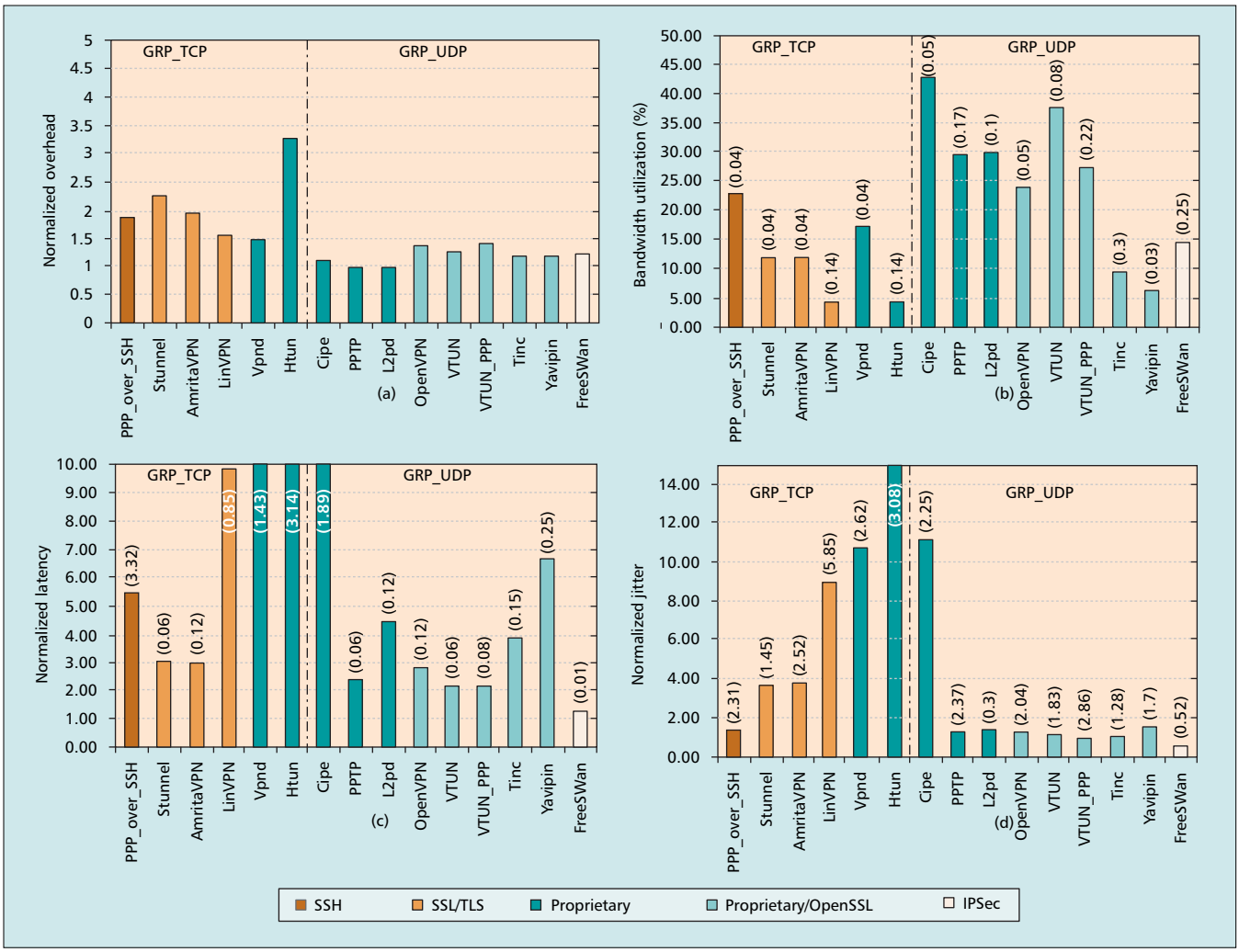


**Figure 3.** *a) Normalized overhead added to every packet; b) bandwidth utilization; c) normalized latency; d) normalized jitter. (The number on top of each bar presents 95 percent confidence interval).*

these protocols as implicitly secure because they have already been subjected to exhaustive peer review. Moreover, security lapses, if any, are widely published over the Internet with ready-made patches. From Fig. 2, *PPP_over_SSH*, *AmritaVPN*, *Stunnel*, *LinVPN*, and *FreeS/WAN* fall in this category.

- ***Proprietary***: Includes OSLV solutions that are based on proprietary security protocols and rated lower than ***open standards***. We analyze the security offered by these protocols in terms of support for basic properties including *confidentiality*, *data integrity*, *authentication/ nonrepudiation*, and *anti-replay protection*. Again, we emphasize that supporting all these properties does not necessarily mean the protocol is secure, and more exhaustive analysis clearly needs to be done. Our evaluation is presented in Table 4. *OpenVPN* and *Tinc* are the only solutions that support all four properties.

***Scalability*** — To compare scalability of OSLV solutions, we assumed a full mesh VPN between $N$ nodes and analyzed the total effort required for updating configuration files, editing routing information, and distributing security keys when tunnels are added/deleted. For a full mesh, it all boils down to creating $(N - 1)$ additional tunnels. Since a central management infrastructure is absent, the configuration files for all $N$ sites may need to be updated with reachability information. Similarly, since most OSLVs do not have an integrated routing daemon, routing information may need to be manually updated at every site. Finally, for distributing security keys, if the OSLV solution uses secret key or public key cryptography without a public key infrastructure, the shared secret key will need to be manually distributed to the $N - 1$ peers. On the other hand, if it uses digital certificates, only the newly added site will need to acquire a signed certificate. Table 5 provides further details about the number of tasks that need to be performed for each OSLV. For example, in *PPP_over_SSH* only one configuration file (~/.ssh/config at the $N$th node) needs to be updated, $2(N - 1)$ routes need to be added (at both ends), and public key of the new node needs to be distributed to its peers (for passwordless login). Total efforts can then be calculated by summing up all the tasks and multiplying it by $N$ (since this has to be repeated $N$ times for an $N$-node VPN). Assigning unit effort for every task, the last column

| | OSLV solutions | Remarks |
|---|---|---|
| Low | Stunnel, AmritaVPN, LinVPN | Uses OpenSSL implementation of SSL/TLS. Hence, new algorithms can be used readily without any changes to the OSLV source. |
| | OpenVPN, Tinc, | OpenSSL EVP API is used to access different ciphers/MACs. |
| Medium | PPP_over_SSH | The OpenSSH implementation of SSH uses a fixed list of ciphers that use the OpenSSL EVP-API. Hence, to use new ciphers the code for OpenSSH needs to be modified. |
| | Cipe, Vpnd | Uses proprietary Blowfish. Thus, new algorithms should be explicitly incorporated in source. |
| | Yavipin, VTUN, VTUN_PPP, | Uses the OpenSSL Crypto functions for some selected ciphers. To use newer algorithms, source code will need to be modified. |
| High | FreeS/WAN | Uses Eric Young's DES. It is difficult to locate where modifications need to be made. |
| | PPTP | Uses only Microsoft P2P encryption. |
| | L2TPd, Htun | No ciphers used. |

**Table 2.** *The complexity of introducing proprietary algorithms.*

| Routing | OSLV solutions | Remarks |
|---|---|---|
| Manual | PPP_over_SSH, Stunnel, L2tpd, Htun | Explicit route command is used. |
| | OpenVPN, LinVPN, Yavipin, VTUN, VTUN_PPP, PPTPd | Routing is configured by shell scripts when the VPN link is established. |
| | Cipe, AmritaVPN, FreeS/WAN, Vpnd | Routing information is included in conf. files. |
| Automatic, single-domain VPNs | Tinc | Supports a built-in routing daemon that establishes a full mesh network |

**Table 3.** *In-built support for routing.*

gives a good estimate of scalability. Observe that every solution is unscalable with total efforts varying as $O(N^2)$. However, for small $N$ (say $N = 10$), one can see the drastic difference in the amount of work required (indicated by numbers in square brackets in the last column). Here, *Tinc* proves the most scalable OSLV solution.

## Conclusions and Future Work

In this article we have evaluated 15 popular OSLVs in terms of network performance, features and functionalities, and operational concerns. The relatively poor performance of OSLVs in almost all considered attributes opens new opportunities for research. One such area is improving network performance. A major bottleneck for poor network performance was found to be the compute-intensive nature of the encryption, authentication, and compression functions (called VPN functions) studiously applied to every packet sent over the VPN. Once these functions and their respective algorithms are negotiated during tunnel initialization, they remain static throughout the lifetime of the tunnel and cannot be changed without resetting it. However, this behavior may be redundant for traffic streams that are already encrypted or compressed. Many applications may also require only a subset of such functions to be applied to their streams; for example, voice over IP calls may need encryption and authentication, but can do without compression. Similarly, not all information steams are equally important. While some may need strong cryptography, others may settle for weaker protocols that are faster and not as compute-intensive. Similar arguments hold for selecting the transport protocol for data channels. Experiments reveal that VPNs using UDP-based data channels are much better than those using TCP. None of the current OSLV solutions are flexible enough to apply such application-spe-

| | Confidentiality | Data-integrity | Authentication/ nonrepudiation | Anti-replay |
|---|---|---|---|---|
| Vpnd | Yes | Yes | Yes | No |
| Htun | No | No | No | No |
| Cipe | Yes | Yes | No | No |
| PPTP | Yes | Yes | No | Yes |
| L2tpd | No | No | No | Yes |
| OpenVPN | Yes | Yes | Yes | Yes |
| VTUN | Yes | No | No | No |
| VTUN_PPP | Yes | No | No | No |
| Tinc | Yes | Yes | Yes | Yes |
| Yavipin | Yes | Yes | No | Yes |

**Table 4.** *VPN security.*

*The results presented in this article can help end-users in selecting an optimum OSLV solution tailored for their specific needs. The simple heuristic algorithm can be extended to accept weighted user requirements and output a ranked list of the best solutions.*

| | Edit configuration files (a) | Update routing information (b) | Distributed secret keys (c) | Total effort (a + b + c) |
|---|---|---|---|---|
| PPP_over_SSH | 1 | $2(N-1)$ | $N-1$ | $N(3N-2)$ [280] |
| Stunnel | $2(N-1)$ | $2(N-1)$ | 1 | $N(4N-3)$ [370] |
| AmritaVPN | $2(N-1)$ | | 1 | $N(2N-1)$ [190] |
| LinVPN | $2(N-1)$ | $2(N-1)$ | 1 | $N(4N-3)$ [370] |
| Vpnd | $2(N-1)$ | | $N-1$ | $N(3N-3)$ [370] |
| Htun | $2(N-1)$ | $2(N-1)$ | 0 | $N(4N-2)$ [480] |
| Cipe | $2(N-1)$ | | | $N(2N-1)$ [190] |
| PPTP | N | $2(N-1)$ | $2(N-1)$ | $N(5N-2)$ [480] |
| L2tpd | $2(N-1)$ | $2(N-1)$ | $(N-1)$ | $N(2.5N)$ [250] |
| OpenVPN | $2(N-1)$ | | 1 | $N(2N-1)$[190] |
| VTUN_* | $2(N-1)$ | | | $N(2(N-1))$ [180] |
| Tinc | 1 | 1 | 2 | $N(4)$ [40] |
| Yavipin | $2(N-1)$ | $2(N-1)$ | 1 | $N(4N-3)$ [370] |
| FreeS/Wan | $N$ | | $N$ | $N(2N)$ [200] |

**Table 5.** *Scalability of OSLVs.*

cific functionality and newer architectures will need to be designed, that allow VPN functions to be applied at the packet level. In this regard, we are proposing a next-generation VPN architecture called Flexi-Tunes that provides a flexible tunnel implementation where within a single VPN tunnel, different VPN functions can be applied to different applications, thus offering differential and customized treatment. Flexi-Tunes empowers applications on end hosts to either specify the kind of treatment they expect for their traffic streams or take active part in applying the tunneling functions themselves.

The results presented in this article can also help end users select an optimum OSLV solution tailored to their specific needs. The simple heuristic algorithm discussed in the results section can be extended to accept weighted user requirements and output a ranked list of the best solutions. As a result, a home user interested in maximum bandwidth may settle for *Cipe* or *Vtun*, while a network administrator who is more interested in security may settle for IPSec. This method, however, requires the performance of each OSLV to be predetermined. Such results are not readily available over the Internet, and our work helps to fill this gap.

## References


[1] R. Venkateswaran, "Virtual Private Networks," *IEEE Potentials*, Mar. 2001.
[2] J. Epplin, "Peeking Under the Hood of SnapGear's uClinux-powered VPN Appliances," LinuxDevices.com, Jan. 2003, http://www.linuxdevices.com/articles/AT3682158384.html
[3] C. J. C. Pena and J. Evans, "Performance Evaluation of Software Virtual Private Networks (VPN)," *Proc. 25th Annual IEEE Conf. Local Comp. Networks*, Nov. 2000, pp. 522–23.
[4] J. P. McGregor and R. B. Lee, "Performance Impact of Data Compression on Virtual Private Network Transactions," *Proc. 25th Annual IEEE Conf. Local Comp. Networks*, 2000, pp. 500–10.
[5] O. Titz, "Why TCP over TCP Is a Bad Idea," http://sites.inka.de/sites/bigred/devel/tcp-tcp.html
[6] H. Lipmaa, "Crypto Hardware Pointers," http://www.tcs.hut.fi/~helger/crypto/link/practice/hardware.html
[7] B. Schneier, "Security in the Real World: How to Evaluate Security Technology," excerpts from a general session presentation at CSI's NetSec Conf., St. Louis, MO, June 1999; http://www.schneier.com/essay-realworld.html
[8] J. De Clercq and O. Paridaens, "Scalability Implications of Virtual Private Networks," *IEEE Commun. Mag.*, vol. 40, no. 5, May 2002, pp. 151–57.
[9] O. Kolesnikov and B. Hatch, "Building Linux Virtual Private Networks," New Riders, 2001.


## Biographies


SHASHANK KHANVILKAR (skhanv1@uic.edu) received a B.E. degree in electronics from Ramrao Adik Institute of Technology, University of Mumbai in 1997 and an M.S. degree in computer engineering from the University of Illinois at Chicago in 2001, where he is currently pursuing a Ph.D. His research interests include multimedia networks and operating systems.

ASHFAQ KHOKHAR (ashfaq@uic.edu) received his B.Sc. in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 1985 his M.S. in computer engineering from Syracuse University in 1989, and his Ph.D. in computer engineering from the University of Southern California in 1993. Since fall 2000 he has served as an associate professor in the Departments of Computer Science and Electrical and Computer Engineering at the University of Illinois at Chicago. He has published over 90 technical papers in refereed conferences and journals in the areas of parallel computing, image procesing, computer vision, and multimedia systems. He was a recipient of the NSF CAREER award in 1998. His paper "Scalable S-to-P Broadcasting in Message Passing MPPs" won the Outstanding Paper Award at the International Conference on Parallel Processing in 1996. His research interests include wireless and sensor networks, multimedia systems, data mining, and high-performance computing.