

Experimental evaluations of Open-Source Linux-based VPN solutions

Shashank Khanvilkar, Ashfaq Khokhar

Department of Electrical and Computer Engineering, University of Illinois at Chicago
{skhanvl, ashfaq}@uic.edu

Abstract—Virtual Private Networks (VPNs) provide a low-cost alternative to leased lines and as such, are becoming increasingly popular among commercial and defense organizations for providing vital inter-office connectivity. Several commercial and open-source VPN products are now available that mainly differ in their capabilities to provide safe and secure services. In this paper, we study fifteen popular Open-source Linux-based VPN solutions (OSLVs) and compare them with respect to network performances (bandwidth, delay and latency/jitter), supported features & functionalities (algorithm plug-ins and routing) and operational concerns (security and scalability). Experiments suggest that there is no single OSLV that excels in all the considered aspects and a combination of different solutions and/or tradeoff among desired characteristics may be required to deliver an optimal performance. Also network performance results suggest that OSLVs using UDP tunnels introduce 50% lower overhead, utilize 80% higher bandwidth and have 40-60% lower latency/jitter than those based on TCP.

Keywords- VPN, performance evaluation

I. INTRODUCTION

The exponential growth in Internet has enabled organizations to expand their operations and globalize their presence. This has prompted development for new techniques that enable secure inter-office connectivity and offer cheaper dial-up services for roaming users. Several solutions have been developed for this purpose, but Internet-based Virtual Private Networks (VPNs) seem to be 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 over the Internet. These tunnels use cryptographic techniques to provide robust security and privacy. Moreover, VPNs completely replace leased lines and eliminate the need for special equipment, thus cutting up to 75% of all operational costs.

Several VPN products (both commercial and free) are now widely available. These products differ from each other mainly in terms of cost and capabilities to provide secure, reliable and efficient services. In recent years, however, Open-Source VPN solutions have attracted a lot of attention from researchers and developers alike. These are breed of software solutions that can provide commercial-grade VPN services using desktops 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 hardwares to extract maximum performance; for example LITE+ and SME550 [1]. Since Linux serves as the underlying operating system, deployment is quick and easy,

circumventing the need for system administrators to learn other proprietary operating systems. All this coupled by a 24 x 7 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 (www.linuxrouter.org), these solutions, in our opinion, have the potential to carve a niche in the VPN market.

In this paper, we study fifteen popular Open-source VPN solutions that are freely available over the Internet and evaluate their performances based on traffic characteristics, supported features/functionality and operational concerns. From this study we seek to highlight common drawbacks that currently exist in OSLVs and underscore future research issues that need to be addressed. Recently, Pena and Evans [2] had carried out a similar study, comparing different VPNs in terms of network throughput and CPU usage over high/low-speed networks. However, we feel that there are many other aspects (like overhead, security, modularity etc.) that characterize a VPN solution and hence warrant comparison. Our work is comprehensive both in terms of the number of OSLV solutions considered (fifteen as opposed to six in [2]) and characteristics compared (seven as opposed to two).

Our study reveals that there is no single OSLV that excels in all considered aspects and selecting an appropriate solution involves several tradeoffs depending on security policies and intended applications. For example, FreeS/WAN demonstrates low latency and jitter, but introduces high overhead and utilizes lesser bandwidth. Thus for situations where overhead and bandwidth are important factors, alternate solutions like Cipe or PPTP may be employed. The performance results related to network parameters such as bandwidth and latency suggest that the OSLVs using UDP tunnels introduce 50% lower overhead, utilize 80% higher bandwidth and have 40-60% lower latency/jitter than those based on TCP. In general, we believe that several factors will need to be carefully evaluated before an OSLV solution is selected for deployment and multiple solutions may need to be integrated to maximize performance.

The rest of the paper is organized as follows. For the sake of completeness, Section II provides an overview of a generic software architecture for a typical OSLV solution, detailing packet treatment inside an OSLV router. Section III provides a classification for the OSLVs based on different security protocols used. Section IV describes the test-bed and analyzes the performance results. The conclusions are presented in Section V.

II. SOFTWARE ARCHITECTURE FOR OSLV ROUTER

Fig. 1 illustrates the software architecture of a typical OSLV router node. It is a modification of the basic TCP/IP network protocol stack with two additional components: the *VPN daemon* and the *Virtual Network Interface (VNI)*. The *VPN daemon* is either a user- or kernel-level process that consists of a *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 with multiple stages to provide encryption, authentication, data-integrity and compression functions. Some of these functions may also be implemented in hardware for enhanced performance. 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, for their data channels have significantly poor network performance.

The Virtual Networking Interface (VNI) is an abstract device created during VPN initialization. To an operating system the VNI appears exactly like any real network interface, with the difference that all packets sent to the VNI, get delivered to the VPN daemon instead. Similarly data written by the VPN daemon gets simulated as a packet arrival event, which is treated in the same way as a packet arrival event on any real networking interface. Thus the VNI acts like a bi-directional pipe between the kernel networking module and the VPN daemon, which serves for tunneling packets. Universal Tun/Tap driver (available for Linux, Solaris and FreeBSD) and pseudo-terminals (available on almost all Unices) are some popular mechanisms, used by OSLVs, to create a VNI.

Fig. 1 also outlines the packet data-path as it travels from one private network to another. There are three distinct stages: (1) The IP Routing Daemon (see Fig. 1), first gets the packet and uses its destination address to determine the next-hop router/egress interface (2) Since the destination will be a private address, the routing daemon will drop this packet unless (a) routing table is updated with information to route it through the VNI, which subsequently hands it to the VPN daemon, or (b) Routing Daemon is modified to recognize a VPN packet and directly hand it to the VPN daemon. The second method will eliminate an extra trip down the protocol stack and reduces the associated latency. However, this requires changes to routing protocol, which is more complicated. (3) The VPN daemon treats all packets as data and subjects them to compression and different cryptographic functions. It then determines the IP address of the peer OSLV router, wraps it in a new header and sends it as any other IP packet.

At the receiver the exact reverse steps take place and extra headers are removed as the packet is finally delivered to the correct destination host.

III. OSLV CLASSIFICATION

We have classified the OSLVs into four groups based on the security protocols employed (see Fig. 2).

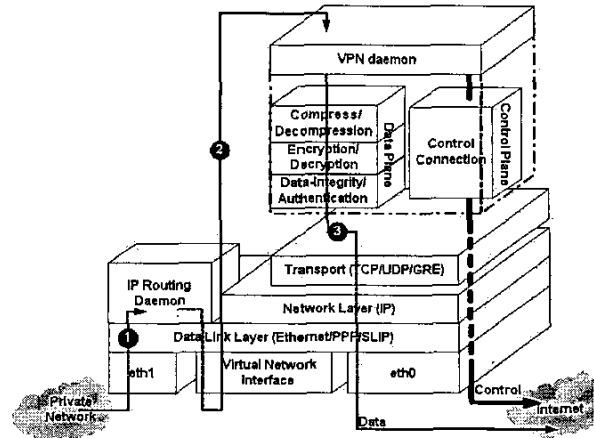


Figure 1. Software Architecture of an OSLV router.

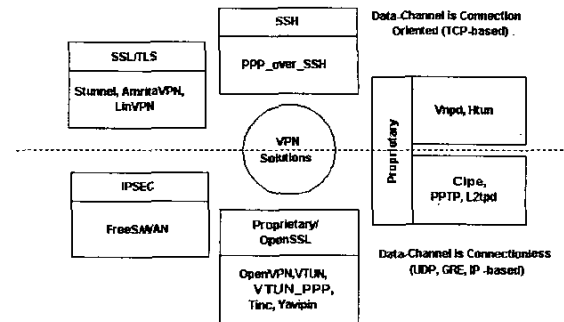


Figure 2. VPN grouping based on Security Protocol.

This classification was used because VPNs with similar security protocols deliver almost similar performances. Other kinds of classifications also exist; for example, Ref. [5] classifies VPNs based on the protocol layer at which it operates. However, since most of the VPNs considered in this paper, operated at the application layer, this classification could not be used here. Thus, we have VPNs based on SSH, SSL/TLS, IPsec and other Proprietary protocols. While SSH, SSL/TLS and IPsec are well-known Internet standards, the last group employs proprietary mechanisms to provide security. All OSLVs in the top half of Fig. 2 (above the dotted line) use TCP data channels, while the rest use UDP.

Secure Shell (SSH), standardized by the *secsh* working group, was designed to support secure logins and file transfers. *PPP_over_SSH* tunnels PPP packets over SSH. Although this solution was not explicitly rated on freshmeat.net, we have included it in this paper because both PPP and SSH are mature protocols with implementations available on almost every platform and a VPN can be set up using a simple command [6].

Secure Sockets Layer (SSL), versions 1 & 2, were initially developed by Netscape to provide secure web transactions. It has recently being standardized by the *tls* working group and renamed as Transport Layer Security, Ver. 1 (TLSv1) or SSLv3. *Stunnel* and *AmritaVPN* use the latest version, while *LinVPN* is still based on SSLv2.

IP Secure (IPSec) extends security at the network level. However IPSEC also supports tunneling mechanism that can be utilized to provide 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 later is built as a user-land application used for negotiating session keys (subsequently used by KLIPS).

The remaining OSLVs use their own proprietary protocols and as such have been placed in the last group. We further divide this group into those that utilize standard cryptographic libraries like OpenSSL (Proprietary/OpenSSL) and those that implement their own cryptographic functions (Proprietary). We feel that this distinction is important as standard libraries guarantee high quality implementations, which cannot be expected from others. *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 both tun/tap driver and pseudo-terminals, we have considered both these organizations as separate OSLVs, with the first one referred to as *VTUN*, and second one as *VTUN_PPP*.

Additional information about the OSLVs has been tabulated in Table. 1. Here, the last column quantifies our experience with set-up complexity (lower “stars” imply simplified set-up), evaluated against some common criteria, involving ease of installation, configuration and support.

IV. RESULTS

Fig. 3 illustrates the test bed used in our experiments. OSLV-A & B are VPN routers that create a secure tunnel between two private networks. OSLV-A is a 2.0 GHz Intel Pentium-4 machine running RH 9.0 with 512 MB of RAM and OSVL-B is a lower end 400 MHz Pentium-II desktop running RH 8.0 with 128 MB of RAM. PC-1 and PC-2 are similar Linux desktops used to carry the network related experiments. The test-bed is isolated (no external congestion) and all links use 100Mbps Fast Ethernet.

TABLE I. OPEN-SOURCE LINUX BASED VPN SOLUTIONS

	Version	Standard	Recipes	Set-up Complexity
PPP over SSH	-	Yes	\$/ssh-vpn.html	*****
Stunnel	4.04	No	\$/ssl-vpn.html	***
AmritaVPN	0.96	No	\$/amvpn.html	***
LinVPN	2.6-pre1	No	\$/linvpn.html	*****
Vpnd	1.1	No	\$/vpnd.html	****
Htun	0.9.5	No	\$/htund.html	****
Cipe	1.5.4	No	\$/cipe.html	**
PPTP	1.2.0	Yes	\$/pptpd.html	****
L2tpd	0.69	Yes	\$/l2tpd.html	***
OpenVPN	1.4.2	No	\$/openvpn.html	**
VTUN	2.6	No	\$/vtund.html	***
VTUN PPP	2.6	No	\$/vtund-ppp.html	****
Tinc	CABAL	No	\$/tinc.html	***
Yavipin	0.9.5	No	\$/yavipind.html	****
FreeS/Wan	2.01	Yes	\$/ipsec.html	*****

(\$=<http://multimedia.ccc.uic.edu/volans>)

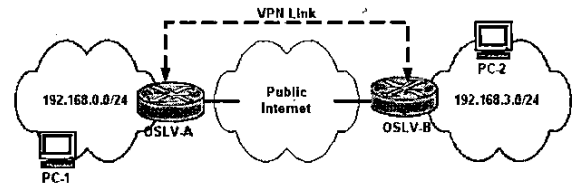


Figure 3. Test bed used for experimentation.

A. Network Performance

To measure network performance, a detailed protocol analysis and verification was carried out to find overhead. *Iperf* was used to measure bandwidth & jitter, while *ping* was used to measure latency. Every experiment was performed 10 times, and only the average has been reported. Since there was not 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.

Fig. 4 illustrates the overhead added by the OSLVs, which ranges from the lowest (~80B added by *Cipe*, *PPTP* or *L2tpd*) to the highest (~220B added by *Htun*). OSLVs using UDP-based data channels (GRP_UDP on the left of Fig. 4) add 50% lesser overhead than those based on TCP. Also OSLVs in the same group are observed to have similar overhead performances. It is clear from Fig. 4 that all OSLVs suffer from large overhead problems and efficient approaches aimed at reducing this overhead need to be investigated. Further analysis of each solution revealed that 75+% of the overhead was due to protocol headers at different layers (see Fig. 1) and the rest was contributed by cryptographic protocols used for security. While it is difficult to reduce the later, compression techniques have been widely used to reduce the former. Compression, however, may not always lead to better network performance [3] and must be used with care.

Bandwidth Utilization of the OSLVs is illustrated in Fig. 5. The numbers on the top of every bar indicates a 95% confidence interval. Again all OSLVs were found to perform poorly, with the best one (i.e. *Cipe*) utilizing <50% and the worst one (i.e. *LinVPN*) utilizing <5% bandwidth. Another key observation is that proprietary protocols utilized higher bandwidth than standard protocols and GRP_UDP outperforms GRP_TCP by at least 80%. This overall lag in bandwidth utilization can be attributed to many factors including, higher overhead, increased latency and TCP stacking. While overhead reduces the amount of useful bytes transferred, higher latency increases the bandwidth-delay product thus requiring larger window sizes. The last factor, TCP stacking, refers to using TCP multiple times along the packet data path causing interference between TCP timers.

Finally performance results related to latency and jitter are illustrated in Fig. 6 and 7, respectively. *FreeS/WAN* is observed to have the lowest latency and jitter characteristics. We believe this happens because *FreeS/WAN* is integrated in the Linux kernel and involves fewer layers of protocols. Here again, GRP_UDP outperforms GRP_TCP by 40 to 60 percent in both cases.

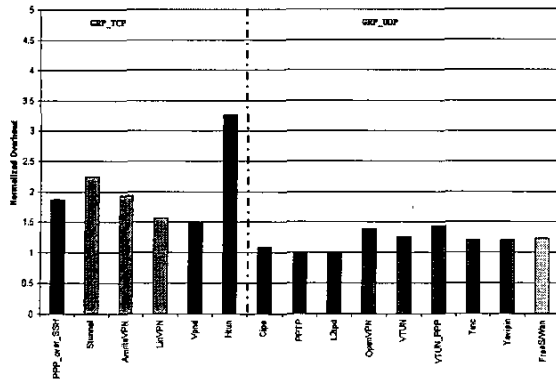


Figure 4. Overhead normalized w.r.t 80 Bytes.

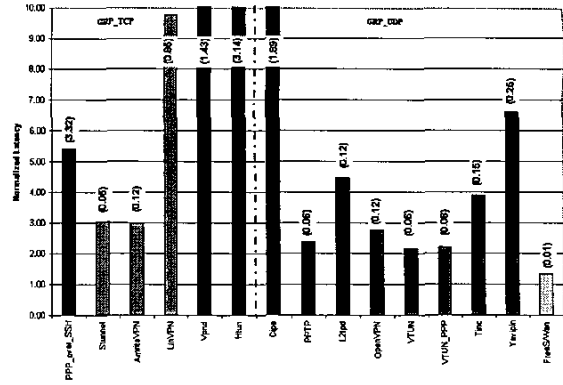


Figure 6. Latency normalized w.r.t min. end-to-end delay.

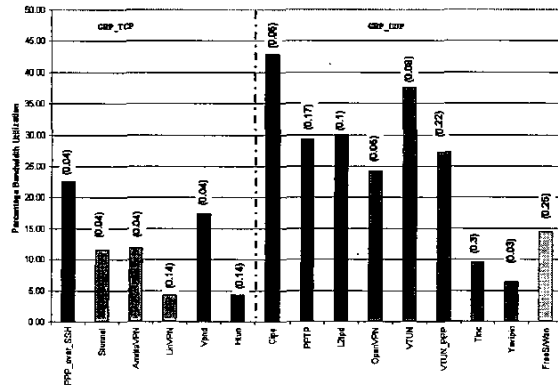


Figure 5. Bandwidth utilization

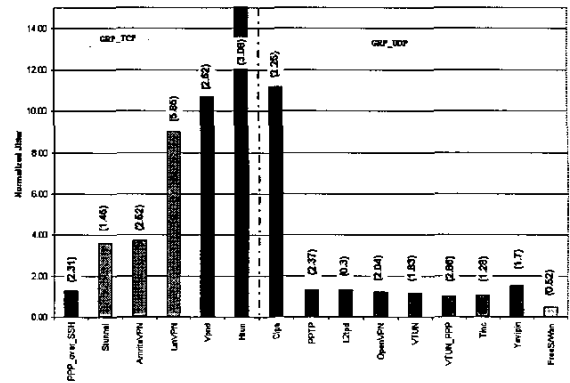


Figure 7. Jitter normalized w.r.t min. end-to-end jitter.

High latencies in OSLVs can be attributed to elongated data path (see Fig. 1), compute-intensive nature of cryptographic functions and multiple copying of data-packets between kernel and user spaces. Layer-2 switching, use of special crypto hardware, and kernel-level integration of VPN Daemons are some of the methods used to reduce this effect. All these methods, however, reduce latency at the cost of flexibility. Similarly, jitter in current OSLVs, can be reduced by using better scheduling and assigning higher priority to the VPN daemon.

From the above network performance results it is difficult to judge the OSLV with the best performance. However, a simple heuristic can be applied to select an optimum solution. For example, by ranking the OSLVs (ranking taken from Fig. 4 through Fig. 7) for each attributes and assigning weights proportional to the importance of that attribute, one can determine the weighted average rank. For example, when all four attributes are given equal weightage, *VTUN*, *PPTP* and *FreeS/WAN* are found to be the most optimum solutions.

B. Supported Features/Functionality

We now discuss two features; built-in routing and code modularity, which when supported considerably enhances the usability of an OSLV.

Built-in Routing: Routing tables needs to be populated once the tunnels are established and the topology is set. The routing tables in every OSLV router need to be updated with information to reach the peer nodes. This can be performed either manually by using the Unix *route* command or automatically using some routing daemon. Manual routing, though simple, presents a cumbersome and error-prone exercise when number of nodes is large. Moreover user-intervention is frequently required to check the status of the connectivity. Automatic routing, on the other hand, can significantly ease this process. For single-domain VPNs, where the address space is unique, a separate routing program like *zebra* can be configured to deliver the routing information. For multi-domain VPNs, however, routing becomes complicated and one possible solution is to have separate routing instances (called virtual routers) to be maintained for every VPN.

Table II. presents our evaluation of the routing mechanisms employed in current OSLVs. We found that manual routing was popular among the OSLVs because of its simplicity. *Tinc* was the only solution that has built-in routing for single-domain VPNs. None of the OSLV solutions, however, support multi-domain VPNs.

Code-modularity: This feature determines the ease with which different algorithms can be used with an OSLV solution. Thus code-modularity can be defined as the OSLV's support

for algorithm plug-ins. For example, there can be a separate plug-in for every cipher and the user can choose only the one that best fits his needs. Plug-ins allow easy integration of newer algorithms and provide the users with an option to create their own or buy independently developed algorithms. Such plug-ins would allow OSLV solutions to survive potential security threats and extend their shelf life. It will also allow companies to use their own proprietary code; a practice that is often discouraged in literature.

TABLE II. IN-BUILT SUPPORT FOR ROUTING.

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 in-built routing daemon that establishes a full mesh network

TABLE III. COMPLEXITY OF INTRODUCING PROPRIETARY ALGORITHMS.

	OSLV Solutions	Remarks
Low	Stunnel, AmritaVPN, LinVPN	Uses OpenSSL implementation of SSL/TLS. Hence new algorithms can be readily used 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 fixed list of ciphers, which 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 algo 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.

None of the OSLV solutions strongly support algorithm plug-ins as defined above. 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 that were used internally. Accordingly, we have divided the difficulty-levels between (see Table III): **Low**: This is the case when the OSLV uses standard 3rd party libraries, like OpenSSL, to implement their security protocols. OpenSSL has a generic interface, called EVP API that provides a standard access mechanism for all its crypto algorithms. Thus newer algorithms can be readily used, with minimal code changes; **Medium**: Difficulty-level is considered medium, when the OSLV requires new algorithms to be manually inserted in the source or it uses older crypto APIs for this purpose. Both cases require the user to hack the

source and figure out the modifications. Note that *PPP_over_SSH* has been included in this category for a special reason (indicated in the remarks column); **High**: This means new algorithms can be used only if substantial changes are made to the source. This occurs if (a) the OSLV did not originally support any crypto functions, which would require starting from scratch or (b) the source code does not provide much help with the modification.

C. Operational Concerns

This section focuses on two attributes: security and scalability, which have important implications that must be clearly understood before settling for a VPN solution. Both are qualitative parameters that are difficult to define and compare. Here we present a number of properties for each of them that might help in knowing them better:

Security: Network Security is an evolving field. There are no explicit definitions for security parameters, which when satisfied, would fully guarantee the security of a system. In other words, there are no litmus tests for security, which makes it difficult, if not impossible, to gauge it. We follow a conservative approach of treating all open-standard protocols to be intrinsically secure and determine basic security support for others. The reasoning behind this logic is that open-standards are subjected to exhaustive peer-reviews and security lapses, if any, are widely published over the Internet with ready-made patches. So any given day, we lay more trust on OSLV solutions that follow open-standards. Thus *PPP_over_SSH*, *AmritaVPN*, *Stunnel*, *LinVPN* and *FreeS/WAN* have preferred status. Proprietary security protocols, on the other hand, are developed by a small community (5 to 10 people) and there is no guarantee that most attacks will always be prevented. Moreover the turn-around time for plugging security holes can be expected to be much larger. Note that exhaustive analysis of the security properties of every proprietary protocol is not feasible. Here, we only determine if such protocols satisfy the following four basic security goals: (a) Confidentiality (b) Data-integrity (c) Authentication/non-repudiation and (d) Anti-replay protection. Our evaluation on the basis of these four goals is presented in Table IV. *OpenVPN* and *Tinc* were the only solutions that satisfy all four goals.

Scalability: Clercq and Paridaens [4] analyze the scalability of provider provisioned VPNs, in terms of *memory consumption*, *processing power* and *configuration and management load (C&M)*. Memory consumption depends on the number of established tunnels. Since every tunnel requires some state to be maintained at each end, 'N' tunnels consume N times the available memory. Similarly processing power is affected by the compute-intensive crypto functions. With higher number of tunnels, the slice of the CPU time allocated for every tunnel is reduced by the same fraction. However, both these factors limit the scalability of only a single node and these problems can be partially eliminated by using powerful desktops with larger processing power and/or larger main memory. Since OSLVs are expected to be used by small-scale businesses (typically 10 to 50 sites), these factors do not play a significant role and can be safely neglected for now. However, for commercial set-top boxes catering to a larger audience, their combined effect will need to be considered.

TABLE IV. SECURITY PROPERTIES FOR PROPRIETARY PROTOCOLS.

	Confidentiality	Data-Integrity	Authentication/Non-Repudiation	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 V. SCALABILITY CALCULATION FOR OSLVs

	a. Conf	b. Rtnng	c. Scrts	T.E (= N*(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]

Here we concentrate on configuration and management load quantified by the total efforts required for updating configuration files (*Conf*), editing routing information (*Rtnng*) and distributing security keys (*Scrts*) when new tunnels are added/deleted. This, in our opinion, affects the scalability of the entire system, which can only be resolved through automation. Thus to evaluate scalability, we consider a VPN between N nodes connected in full-mesh and calculate the total efforts required for *Conf*, *Rtnng* and *Scrts*, when the Nth node is added. The consolidated results for all OSLVs are presented in Table V. For a full-mesh, this boils down to creating (N-1) additional tunnels. In the absence of any central management, configuration files at all N sites may need to be updated and routing information may need to be added to (N-1) nodes for manual and 1 node for automatic routing. Finally for *Scrts*, if the OSLV uses secret-key or Public Key without a public-key infrastructure then the shared secret will need to be manually distributed to its (N-1) peers, but on the other hand, if it uses digital certificates, then only the newly added site will need to acquire a signed certificate. For example, in case of *PPP_over_SSH*, only 1 configuration file (*~/ssh/config* at the Nth 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 password-less login). Total efforts (T.E) can then be calculated by summing up all the tasks and multiplying it by 'N', since this has to be repeated N times. Assigning unit effort for every task, the last column gives a good estimate of scalability. Observe that every solution is unscalable with total efforts varying as O(N²). For small N (say

N=10), one can see the drastic difference in the amount of work required (indicated by numbers on the square bracket). Here, *Tinc* proves to be the most scalable.

V. CONCLUSION

In this paper we have evaluated 15 popular OSLVs in terms of network performance, features & functionalities and operational concerns. The relatively poor performance of OSLVs in almost all considered attributes provides new opportunity for research. For example, one such area is network performance. A major contributor to poor network performance is the compute intensive nature of the crypto/compression algorithms (called VPN functions) and it may NOT be improved by designing faster hardware or better algorithms alone. A more scalable approach may involve selectively applying such functions to fewer packets. For example, a traffic stream that is already encrypted/compressed does not need this additional functionality to be applied again. Moreover many applications may not require all VPN functions to be applied to their streams. Some applications like Voice-over-IP, may require encryption/authentication, but can do without compression. Similarly not all information streams are equally important. While some may need strong cryptography, others may settle for weaker protocols that are not as compute intensive and are faster. Similar arguments hold for selecting the transport protocol for data-channels. Experiments reveal that VPNs using UDP-based Data-channels introduce 50% lower overhead, utilize 80% higher bandwidth and have 40 to 60% lower latency/jitter than those using TCP. This clearly motivates the need for using UDP whenever possible. We are in the process of proposing a next generation VPN architecture (called Flexi-Tunes) that provides such flexible implementation.

The results presented in this paper can also help end-users in selecting an optimum OSLV solution tailored for their specific needs. The simple heuristic algorithm (discussed in section IV) 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 performances of each OSLV to be pre-determined. Such results are not readily available over the Internet and our work helps to fill this gap.

REFERENCES

- [1] J. Epplin, "Peeking under the hood of SnapGear's uClinux-powered VPN appliances", LinuxDevices.com, Jan 2003.
- [2] C. J. C. Pena, and J. Evans, "Performance Evaluation of Software Virtual Private Networks (VPN)", Proc of 25th IEEE Conf. on LCN, pp: 522-523, Nov 2000.
- [3] J.P. McGregor and R.B. Lee, "Performance impact of data compression on virtual private network transactions", Proc. of 25th IEEE Conf. on LCN, pp: 500-510, 2000.
- [4] J. De Clercq and O. Paridaens, "Scalability implications of virtual private networks", IEEE Comm. Mag, May 2002.
- [5] S. Patton, B. Smith, D. Doss, W. Yurcik, "A layered framework strategy for deploying high assurance VPNs", 5th IEEE Intl. Symp. on High Assurance Sys. Engg., pp:199-202, 2000.
- [6] O. Kolesnikov and B. Hatch, "Building Linux Virtual Private Networks", New Riders, 2001.