Vortex: Enabling Cooperative Selective Wormholing for Network Security Systems

John R. Lange, Peter A. Dinda, and Fabián E. Bustamante

Northwestern University, Evanston IL 60208, USA
{jarusl,pdinda,fabianb}@cs.northwestern.edu

Abstract. We present a novel approach to remote traffic aggregation for Network Intrusion Detection Systems (NIDS) called *Cooperative Selective Wormholing* (CSW). Our approach works by selectively aggregating traffic bound for unused network ports on a volunteer's commodity PC. CSW could enable NIDS operators to cheaply and efficiently monitor large distributed portions of the Internet, something they are currently incapable of. Based on a study of several hundred hosts in a university network, we posit that there is sufficient heterogeneity in hosts' network service configurations to achieve a high degree of network coverage by re-using *unused port space* on client machines. We demonstrate *Vortex*, a proof-of-concept CSW implementation that runs on a wide range of commodity PCs (Unix and Windows). Our experiments show that Vortex can selectively aggregate traffic to a virtual machine backend, effectively allowing two machines to share the same IP address transparently. We close with a discussion of the basic requirements for a large-scale CSW deployment.

Keywords: wormholes, honeynets, honeypots, volunteer systems.

1 Introduction

We present *Cooperative Selective Wormholing (CSW)*, a novel approach to providing traffic for use in network intrusion detection systems (NIDS). Our approach adopts a cooperative model [8,11,23,24] in which volunteers contribute their hosts' unused network ports and a portion of their bandwidth. NIDS operators selectively aggregate the traffic bound for these ports in order to effectively monitor large distributed portions of the Internet.

Collecting and analyzing network traffic to detect new methods of attack has long been recognized as a necessity by the security community, and numerous systems have been developed to provide such a service. While the design and functionality of these systems are vastly different, nearly all of them operate by aggregating network traffic from some source. CSW is such a source, one whose volunteer nature presents the potential for dramatically improved coverage of the Internet.

CSW is inspired by the wormholing model of Weaver, et al. [30], in which a dedicated, low-cost hardware frontend device is attached to a network of interest

C. Kruegel, R. Lippmann, and A. Clark (Eds.): RAID 2007, LNCS 4637, pp. 317–336, 2007.

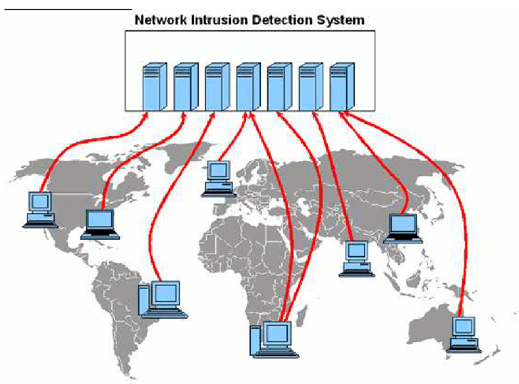


Fig. 1. Cooperative Selective Wormholing provides distributed traffic aggregation for NIDS through volunteer PCs

to forward traffic for a range of IP addresses to a backend honeypot. CSW also uses a frontend/backend distinction, but it does not require any hardware deployment and allows individual machine owners to participate. CSW enables the aggregation of traffic bound to specific unused ports, thus allowing the wormhole to transparently coexist on a volunteer machine. Figure 1 illustrates CSW at a high level.

Network telescopes are the currently preferred method of traffic aggregation in the security community [17]. By providing access to portions of the routed IP address spaces on which little or no legitimate traffic exists, network telescopes make possible the monitoring of unexpected network events such as network scanning or some forms of flooding DoS attacks. Perhaps the main drawback of the telescope approach is that it inherently restricts access to network traffic to well-connected or well-funded individuals or groups capable of convincing an organization to redirect its traffic to a remote location.

While it has been shown that accommodations for network telescopes can be made, the model creates barriers for unaffiliated and unconnected investigators. In contrast, CSW makes it possible for *any* researcher to deploy a large scale distributed traffic aggregation infrastructure, solely by finding *individual* volunteers. It has clearly been shown that it is possible to convince individuals to volunteer resources for a research effort, often on a massive scale [8,11,23,24]. CSW is similar to such efforts, except that individuals volunteer *unused ports and bandwidth*.

CSW wormholes capture traffic destined for unused ports on the volunteer's machine and tunnel it to generic backend NIDS that are stood up by researchers and others. The sender of the traffic is ideally completely unaware that he is in fact interacting with a backend instead of with the volunteer's machine. Furthermore, the wormhole only runs on unused ports, none of the volunteer's own traffic is disclosed to the backend, alleviating privacy concerns.

As a first step to realizing the CSW vision, we have developed *Vortex*, a prototype tool that enables volunteers to instantiate cooperative selective wormholes on their machines. Vortex was developed based on our experience with network virtualization and high performance grid computing. It is implemented using VTL and VNET, two toolsets we have presented previously [10,25]. Vortex runs on both Unix and Windows environments without interfering with any local activity. Our evaluation of Vortex helps to establish the feasibility of CSW and acts as a corner stone to its implementation.

We now elaborate on the three central issues of CSW:

- Coverage: Does the Internet possess enough diversity in open network port configurations to provide acceptable amounts of traffic for CSW?
- Invisibility to clients: Can CSW systems be designed in such a way as to not inconvenience the user running them?
- Invisibility to attackers: Will attackers be able to detect the presence of a CSW on a volunteer's machine?

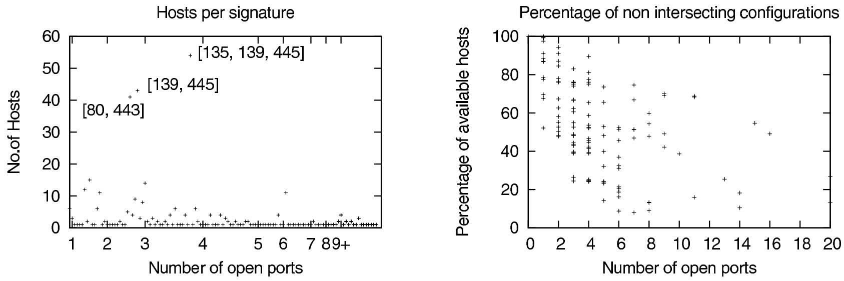
We will also present the design, implementation, and evaluation of Vortex, our CSW proof-of-concept tool.

2 Coverage

The principal issue with any traffic aggregation technique is the degree of coverage it can obtain over the network. We define coverage as the distribution of traffic aggregators over a sample space. For instance, a network telescope gives very fine-grained coverage over a very small area, so it can very accurately capture behavior inside a subnet, but it cannot accurately describe the Internet at large. Until now the distribution of monitored *addresses*, the *horizontal coverage*, was the only coverage that needed to be considered. CSW improves *horizontal coverage* by allowing cheap access to more widely distributed addresses, however CSW has its own coverage issue: can a CSW system cover a relevant sample of network ports? We use *vertical coverage* to denote coverage of network ports.

2.1 Horizontal Coverage

The largest advantage of CSW systems is the possibility of gaining a large degree of random coverage over the entire Internet address space. Telescopes inherently sample at a very low resolution, on the order of large subnets, and are difficult to deploy remotely, so their localized observations may not be representative of the actual activity taking place across the entire Internet. On the other hand, CSW systems could provide a *random sample* of the Internet address space, thus ensuring more widely applicable analyses. Note that the utility of using volunteer hosts to gain a distributed presence has already been established [21].



(a) Hosts per Signature (b) Signature Availability

Fig. 2. Signature prevalence and intersections from random sample of university hosts

2.2 Vertical Coverage

While CSW systems have the potential to provide superior resolution at the address level, they inherently restrict port coverage. The issue of this *vertical coverage* is specific to CSW, since the resolution possible is dependent on the heterogeneity of the available ports, and their combinations, present in the Internet. This heterogeneity corresponds to both the prevalence of operating systems as well as the diversity of services active on volunteer hosts.

A CSW system would ideally be able to stand up port configurations in the same proportion that those configurations are present in the actual Internet. If some percentage of machines run a web server then the wormholed traffic would ideally represent that same percentage. Thus CSW loses its appeal when too many potential volunteer clients share the same open port configuration.

To gain an idea of the port distribution in the Internet we conducted a study of the Northwestern University network. We scanned the first 1024 TCP ports over 1000 machines randomly selected from the university network. We then culled invalid devices (network switches, printers, etc) from the results, and analyzed the remaining scans to identify the distribution of open ports. We refer to the set of open ports on a machine as the *port signature* or simply the *signature* of that host.

We positively identified 401 of the 1000 addresses as capable of operating as a Vortex sensor (general purpose computers running a Vortex-compatible OS) using the OS fingerprinting functionality of nmap. An additional 253 of the 1000 hosts had no open ports, suggesting either an unknown OS, or (more likely) a firewalled/secured OS configurations. It is reasonable to believe that many of these additional hosts are also Vortex-compatible. Nonetheless, we focus on the (worst case) 401 machines, from which we detected 123 distinct port signatures.

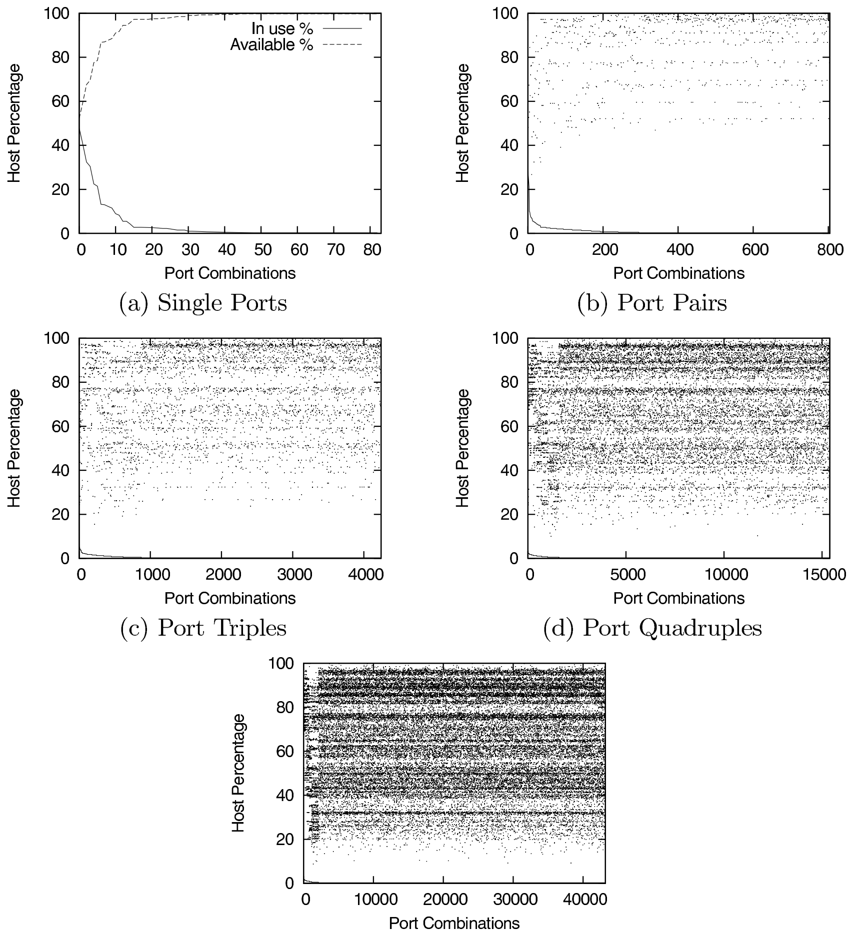
Configuration Diversity. We analyzed the number of hosts running each signature, to determine if there were any obviously prevalent signatures. The results are are shown in Figure 2(a), which plots the number of relevant hosts as a function of the size of the port signature. There are three non-surprising prevalent signatures, which we label. The most popular signature includes three ports associated with standard Windows services commonly present on machines acting as Windows file servers. The second most popular signature contains a subset of the Windows file server ports, consistent with a standard Windows desktop machine. Finally, the third most popular signature consists of ports used by web servers (http and https). Taken together these signatures are present on 138 (34%)of the 401 hosts we scanned. The figure also shows that there are 81 hosts (20%) with signatures that are unique to our data set, suggesting that there is a significant degree of diversity in port signatures. The remaining 45% of hosts exhibited a diverse range of signatures.

Configuration Separation or Non-Intersection. If a selection of signatures all included a common subset of ports, the effectiveness of CSW in that selection would be greatly diminished. We define any host whose port signature does not intersect a given configuration that we want to monitor as an available host for that configuration. To analyze the degree of separation present in various signatures, and the availability implications, we used three approaches.

Entire Signatures. We first determined the amount of intersection among the signatures themselves. We considered each signature in turn and determined the number of hosts in the set that did not intersect with the selected signature. These non-intersecting hosts would be available for a CSW of the prospective signature. The results are shown in Figure 2(b). We can see that the signatures with the fewest open ports are also the signatures with the highest degree of availability. This is especially notable when considered with the previous observation that the most prevalent signatures had only a small number (2 or 3) ports. The *minimum* availabilities of 2 and 3 port signatures are 50% and 20% respectively. More importantly, as the number of open ports increases the number of available hosts does not decrease to zero. The worst availability is 7.98% (32 hosts). The typical availability is much higher. Also, we are likely underestimating availability as we are not counting hosts that have no open ports.

Port Combinations. We also measured the separation of the signatures by considering subsets of the ports included in each signature. We considered each unique signature and looked at the combinations possible when selecting a given number of ports from the signature. Figure 3 shows the availability of combinatorial results obtained from choosing different numbers of ports ranging from 1 to 5. When choosing a single port (Figure 3(a)), the availability is simply 100% minus the percentage of hosts using that port. The top line represents the availability of the single port while the bottom line shows the percentage of hosts containing that port in their signature. In the rest of graphs we retain the bottom line showing the percentage of hosts containing the port combination in their signature, but use a scatter plot to show the availability. Each point represents one of the possible port subsets of the given size. The results are sorted by port combination popularity, with the most common port combinations to the left.

While our previous results on the availability of entire signatures lets us estimate an upper limit for each signature, the combinatorial analysis we describe here is trying to isolate common port combinations present in the trace, letting us estimate a lower limit. Figure 2(a) shows that there are three popular small



(a) Single Ports (b) Port Pairs (c) Port Triples (d) Port Quadruples (e) Port Quintuples

Fig. 3. Probabilities that a given port combination is available in the set of hosts gathered from the randomized port scan. Port combinations are calculated from the signatures present in the scan results, and sorted by decreasing popularity.

port signatures, however combinations including those popular ports are likely more common. For instance, if a given machine M has the port signature <80, 139, 443, 445> then it would *not* be included in the host count for either the <80, 443> or <139, 445> signature, even though it would not be available for either signature. By analyzing port combinations we can treat machine M as belonging to both the <80, 443> and <139, 445> signatures.

Interestingly the graphs for the four higher order combinations (Figure 3(b)-(e)) exhibit a common structure. Each contains bands of availability in roughly the same locations, as well as a common dip in availability for combinations of medium popularity. The other notable aspect of the graphs is that there is a

Tag	*MachineClass*	Signature (Open Ports)
EXG	Exchange Server	25, 80, 110, 135, 137, 139, 143, 443, 445, 593, 691, 993, 995
LHS	Linux Hosting Service	22, 25, 80, 443, 993, 995
WIN	Windows Desktop	139, 445
WFS	Windows File Server	135, 137, 139, 445
LMS	Linux Mail Server	22, 25, 119, 515, 635, 993
LWS	Linux Web Server	22, 25, 80, 443
WDC	Windows PDC	53, 88, 135, 139, 389, 445, 464, 593, 636
SMB	Linux Mail + SMB	22, 25, 119, 137, 138, 139, 445, 515, 631, 993

Fig. 4. Partial list of Common Machine Configurations and their Signatures

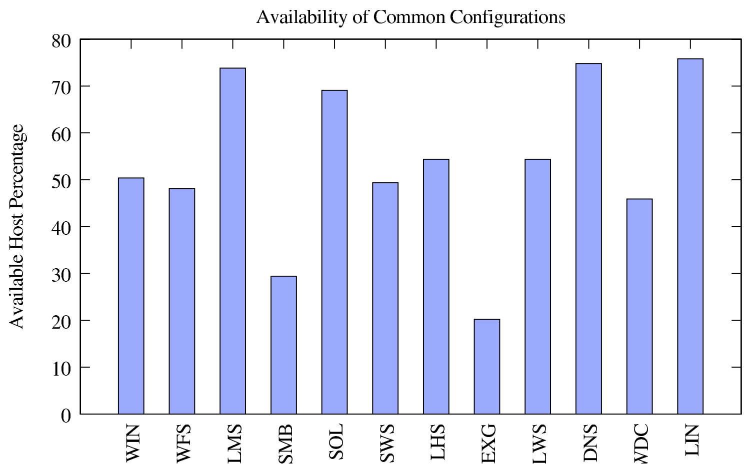


Fig. 5. Host availability for selective wormholing of several common signatures. Availability is measured as the percentage of hosts in the host set returned from a randomized scan that are available for a signature.

more concentrated collection of availability bands towards the top, indicating that there is substantial availability for a large subset of port combinations.

The upshot of this analysis is that it provides considerable confidence that even with a restrictive definition for port signature intersection, very popular port signatures are likely to be available on a substantial number of Internet hosts.

Signatures of Common Machine Configurations. We next consider the availability of port signatures found on currently common machine configurations. Figure 4 contains a subset of the configurations and signatures we tested. We included common operating systems (including Windows, Linux, Solaris, and MacOS X), as well common configurations of those operating systems (such as web servers, email server, and domain controllers). We considered the signature for each common configuration and ran it against our host set to determine the availability of the configuration.

Figure 5 contains the results for the configurations we analyzed. Each machine configuration is shown along with its corresponding availability in our host set. The graph shows that the availability is quite good for most common configurations. The worst cases are machines configured as Windows Exchange servers as well as a Linux mail servers running Samba. This is to be expected since both of these signatures contain standard Windows ports as well as ports commonly found on Unix and server-class machines, effectively bridging the different machine configurations. Still, the results show that at least 20% of hosts will always be available for any of the given machine configurations.

2.3 Coverage Feasibility

The results from the analysis of our random sample of machines connected to the Northwestern University network show that there is a substantial amount of heterogeneity present. While our results may be somewhat limited to our specific environment, they indicate that the feasibility of obtaining vertical coverage in the Internet as a whole is likely substantial. Our results suggest that CSW is likely to be an effective method for collecting traffic from a large and statistically meaningful sample of the Internet. We should also note that our analysis does not take into account intermediate network devices such as NATs and firewalls. If such devices are present then they would interfere with any active CSW located behind them. We currently do not address these intermediate devices other than to say that mechanisms, such as UPNP, do exist that could possibly allow traffic to be delivered to a CSW through a NAT or firewall.

3 The Vortex Cooperative Selective Wormhole

To provide a proof-of-concept and to study other aspects of CSW, we have developed Vortex. Vortex interfaces with an overlay networking system to support connectivity with different backends. Vortex is an outgrowth of research into using virtual machines (VMs) for high performance distributed computing, and so is built using several tools developed in that work. While these tools provided a general and easy avenue for implementing a CSW they are by no means required. Vortex could, for example, be easily implemented as a firewall extension.

3.1 Design

Vortex functions by instantiating a CSW on a client machine and communicating with a VM running as the backend system. This configuration is what would typically be seen if Vortex was used as a traffic aggregator for a virtual honeypot system. Vortex was implemented using our VTL and VNET toolsets [10,25]. Although we evaluate Vortex with a VM backend, the generality of VNET allows a wide range of backend systems to be used, such as passive monitors, monitors that perform simple connection interaction, virtual honeypots, or even physical honeypots. The Vortex architecture is illustrated in Figure 6.

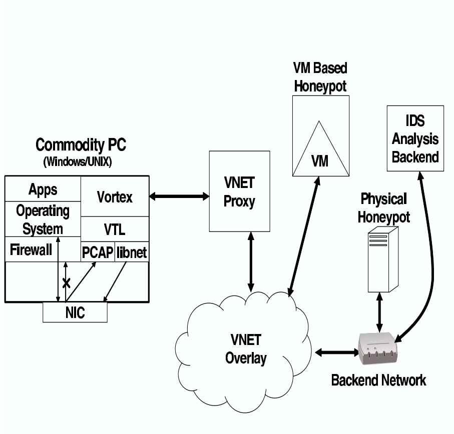


Fig. 6. Vortex Architecture. Vortex uses VTL to capture packets before they are dropped by the host firewall. The captured Ethernet frames are then sent to a VNET proxy which routes the traffic to an IDS backend system.

VTL is a framework designed to allow developers to rapidly develop transparent network services [10]. Primarily, it provides OS-independent methods for packet serialization, acquisition, and manipulation as well as state models used when working with stateful connections. VTL is built on top of Pcap [13,31] and libnet [12], thus providing a cross platform method for interacting with network traffic. Vortex uses VTL for both selective traffic capture as well as transmission of any outbound traffic from the VM backend. Vortex also relies on the VTL mechanisms for packet modification to ensure that traffic is accepted by all parties as legitimate.

VNET is an overlay network toolkit designed specifically for virtual machine-based environments. It provides a layer 2 abstraction for the VMs, tunneling complete Ethernet frames through an overlay whose topology and routing rules are globally controlled [25,26]. Vortex is designed to interface with VNET to provide connectivity to the virtual machine backend. At startup Vortex connects to a VNET proxy machine that routes all the traffic from the wormhole to a specific VNET-connected VM. Because VNET encapsulates entire Ethernet frames, traffic can move seamlessly between the overlay and a physical network interface. This allows VNET to connect to physical network devices as well as virtual network devices exposed by a VM.

The current version of Vortex has a very simple interaction model. A Vortex client instantiates wormholes on any number of unused ports and forwards all traffic to wormholed ports to a single VNET proxy. The VNET overlay is then configured to route all traffic from a given wormhole to a single VM. The VM is configured with the same IP address and routing table as the client machine, but has a separate MAC address. Any traffic that is generated by the backend VM on a wormholed port is tunneled back to the client where it is injected into the physical network. Despite the simplicity of the current interaction model, creation of more complicated use models is entirely possible within the VTL+VNET framework. For example, different wormholed ports on the same frontend could be routed to separate backend systems. Also, more stringent requirements on the traffic generated by the backend system could easily be added. The Vortex client can also perform any number of packet transformations, at layers 2 through 4, to traffic passing both in and out of the wormhole.

We chose to use the VNET+VTL architecture over more established tunnelling architectures, such as GRE, due to the ease of integration, packet access capabilities, and cross platform availability.

3.2 Wormhole Cloaking

While VTL and VNET handle the transmission of network packets from the volunteer machine to a backend system, Vortex itself must assure seamless integration of the backend with the client and the client's network. In order to transparently instantiate wormholes on a volunteer machine, Vortex must fool not only the outside world but also the local machine into handling packets as if they were generated locally. To operate transparently, any packets that are transmitted out of the wormhole must appear as if they were generated by the local machine. Also, if a particular port is being wormholed the local host must not reply to any traffic it receives on that port. Furthermore, in the case of a honeypot backend, traffic must be modified so that it is accepted by the honeypot. We now consider two key issues that must be addressed.

MAC Addresses. This issue only arises when a backend system wishes to interact with traffic that has been captured, that is it wants to send responses and receive replies. In this case any packets generated by the backend would need to share the same MAC address as the volunteer machine. It is feasible that the volunteer could report the MAC address of their machine and require the backend to configure itself to assume that address itself. However, this would require assumptions about the aggregation technique to be made by the backend, something we try to avoid. Also it would require volunteers to divulge information about themselves, which we also seek to avoid.

Instead of requiring the backend to handle this issue we instead have Vortex perform MAC address translation locally on the volunteer machine. Vortex first probes the local machine for its MAC and IP addresses, and then issues an ARP request through VNET to the backend for the local host's IP. The backend responds with an ARP reply containing its MAC address, which Vortex intercepts and stores. From that point onward Vortex rewrites incoming packets with the appropriate MAC address before forwarding them to either the backend or the local network. To ensure ARP table consistency all ARP requests and replies received by the local machine are captured by Vortex and sent to the backend, similarly any ARP packets generated by the backend are inserted into the clients local network.

Packet Suppression. The normal response of a TCP stack to a packet arriving on a closed port is for a host to send an RST packet to the source. However, in the case that the non-open port is being handled by Vortex, this behavior is unacceptable. The result would be a source host receiving both a RST packet

```
-A VORTEX_FW -p tcp –dport 6000:6050 -j ACCEPT
-A VORTEX_FW -p udp -m udp –dport 137 -j ACCEPT
-A VORTEX_FW -p udp -m udp –dport 138 -j ACCEPT
-A VORTEX_FW -m state –state ESTABLISHED,RELATED -j ACCEPT
-A VORTEX_FW -j DROP
```

Fig. 7. Example firewall (IPTables) rules to enable packet suppression

as well as whatever response was generated by the backend for every packet sent through the wormhole. The additional RST would not only likely interfere with the TCP connection, but it would also make Vortex's existence obvious if the source were an attacker. For Vortex to function correctly these RST packets must be suppressed.

To handle the TCP RST problem we use the local host firewalls included in most current OS environments, e.g. iptables and the Windows Firewall. These firewalls support configurations that simply drop packets destined for a port disallowed in the firewall rules, thus ensuring that packets to a closed port never reach the local TCP stack (this is the default behavior for the Windows Firewall). Figure 7 includes an example configuration for iptables. The example accepts TCP packets destined for local ports 6000-6050, udp packets for ports 137 and 138, and packets belonging to an established connection. All packets not included in the rules are dropped and never reach the local TCP stack. Because Vortex has no mechanism capable of blocking the local client from either receiving or transmitting packets, Vortex requires such firewall configurations to be in place in order to operate transparently. This requisite relationship between Vortex and local firewalls leads us to believe that CSW might be best implemented as a firewall extension.

4 Invisibility to Volunteers

A core requirement for Vortex is that it be able to function on a volunteer machine without any interference or impact on performance. While Section 3 discussed the mechanisms required to make Vortex traffic indistinguishable from normal host traffic, those are merely the basic requirements for CSW systems to function. In order for CSW systems to be effective they must also be invisible in more subtle ways, such as in their performance impacts or interference with applications the host machine is running. This requires that CSW systems implement mechanisms for detecting user behavior and reacting accordingly. The current experimental version of Vortex does not implement all of these mechanisms, but we envision incorporating them into a later version. To be truly invisible to a volunteer, a CSW must address the following issues.

4.1 Port Collisions

The most obvious form of interference from Vortex arises from port collisions. This occurs when a wormhole and a local application are simultaneously communicating on the same port number. This can happen when Vortex is launched or if Vortex has been configured on a specific port, when at some later point a local connection begins using that port. Because Vortex maintains transparency to the local machine, it must be able to detect these events on its own and close the wormhole if such an event occurs.

A client's list of open ports is readily available via the **/proc** directory on Linux and a win32 API call in Windows. Currently, Vortex employs active polling of the corresponding mechanism to acquire a list of open ports on the local machine and detect collisions with any active wormholes. If a collision is detected then Vortex closes the wormhole immediately. Polling is neither efficient nor responsive, so we plan on implementing a method for notification when the local host requests the use of a currently wormholed port. We plan on utilizing an NDIS driver hook for Windows environments and library interposition for Unix.

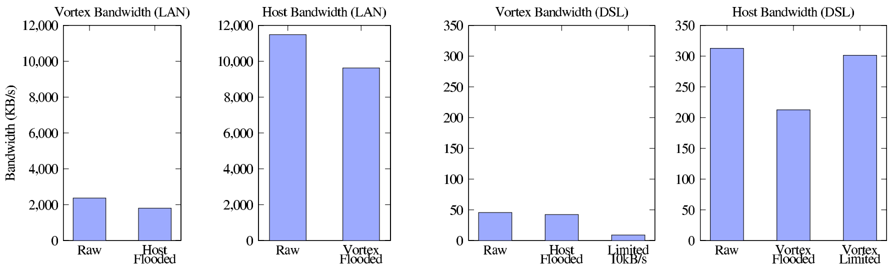
4.2 Performance Degradation

A CSW implementation must also ensure that performance does not significantly degrade on the client machine. This is especially critical for deployments relying on the use of volunteers, as no one will run a tool that slows their machine down noticeably. The performance impact can either be in the form of bandwidth usage or CPU utilization for packet processing.

To address this issue we are working on mechanisms that allow a user to specify the amount of resources they are willing to make available. This technique has been used successfully in many peer-to-peer applications as well as in cooperative computing initiatives. Our plans with Vortex include the implementation of a bandwidth rate limiter that is configurable by the user. This will allow a volunteer to determine the amount of bandwidth which they are willing to provide to Vortex. Another solution could include rulesets for when CSWs are allowed to be instantiated, for example only after a machine has been idle for a given amount time.

Network Overheads. We ran a series of experiments to quantify the performance of a CSW system (Vortex) as well as the possible performance impact on the client machine. Our experimental setup consisted of a Vortex client connected to a virtual machine backend located on the Northwestern University network. We ran two sets of experiments: first with the client located on a home network connected to the Internet via DSL, and a second with the client on the Northwestern network. In each case the traffic to the client was generated from machines on the Northwestern network. For each experiment we measured the raw bandwidth of both the client machine and the wormhole, as well as the available bandwidth of both when the other is being flooded with network traffic. We also used the Linux IProute2 implementation to test the impact of a bandwidth limiter.

Figures 8(a) and 8(c) illustrate the performance of a Vortex CSW. The graphs show the bandwidth through a wormhole under various conditions. Figure 8(a) shows the bandwidth of a Vortex CSW running on a client connected to the same LAN as the backend system, while Figure 8(c) shows the same for a client located on a DSL line. The *Raw* column shows the maximum available bandwidth



(a) Vortex LAN (b) Host LAN (c) Vortex DSL (d) Host DSL

Fig. 8. Bandwidth measurements of clients hosted on DSL and LAN connections. (a) and (c) contain the performance results of a Vortex CSW, while (b) and (d) contain the performance results of the client machine. Measurements were taken under various traffic conditions and bandwidth is measured in kilobytes/second. Each figure shows the mean taken from 10 separate trials.

to a CSW when no other host traffic is present. Vortex is implemented in user space requiring two context switches for each packet received (One to receive the packet from PCAP or VNET, plus one to transmit the packet with libnet or VNET). While these context switches place a limit on the performance of the present version of Vortex, a more intelligent in-kernel design would be capable of performing much better. The figures also show the performance impact on the CSW when the client host is being flooded with traffic to a local service. In both cases Vortex is not starved of bandwidth and continues to function. Finally, for the DSL client we configured the kernel to limit the bandwidth from the client to the VNET proxy, which we will discuss later.

Figures 8(b) and 8(d) show the performance of the volunteer machine running a Vortex CSW. We performed the same experiments as described earlier. First we measured the *Raw* bandwidth of each client machine with Vortex not running and no other traffic present. We then performed the same test but with a Vortex CSW configured and being flooded with traffic. Both hosts show a drop of performance when Vortex is being used heavily, but our implementation is able to cushion the performance drop to roughly 15% due to its architecture. However, as we stated above, the performance of a CSW can be improved, and any improvement will adversely effect the performance of a host. There is clearly a tradeoff that the volunteer needs to make.

To demonstrate that constraints can be placed on a CSW system to prevent it from causing too large a drop in host performance, we ran the tests again with an external bandwidth limiter. For this experiment, we only measured the performance of the DSL host. In order to constrain Vortex we configured a network queue, using IPRoute2 in the Linux kernel, to limit any traffic to the VNET proxy to 10kB/s. We then measured the performance of both the Vortex CSW as well as the client machine. The results are included in the third column of figures 8(c) and 8(d). Figure 8(c) shows that the bandwidth of the CSW is indeed constrained to 10KB/s, while figure 8(d) shows that the performance degradation on the host is limited to only 10KB/s. By incorporating a user configurable limiter with a CSW implementation, volunteers will be able to decide the amount of bandwidth they are willing to donate and be assured that the wormhole won't monopolize their host or network.

4.3 Privacy Risks

Dealing with privacy in distributed network monitors has been recognized as a key concern for any system to be deployed [4,15]. To protect volunteers as well as to minimize the liability of wormhole operators, selective wormholes must be very careful about what traffic they allow to be aggregated. The most serious privacy issue that a CSW must deal with is ensuring that no private local traffic is mistakenly aggregated, but other smaller issues exist as well. For a CSW architecture to be successful it must alleviate any concerns that the volunteers might have.

There is no perfect way to address the problem of mistakenly aggregating private traffic. Ultimately the issue is tied to the behavior of the user and the other members of the users network. For instance, if a volunteer provides a Linux client on a corporate LAN, Vortex could be instructed to instantiate wormholes associated with windows file sharing services. If for some reason another user on the LAN decides to start trying to communicate with those ports, then traffic is being aggregated that might possibly be very sensitive in nature. In other words, a CSW system like Vortex can do nothing to prevent users from purposefully but mistakenly transmitting sensitive information through a wormhole. Vortex's method of preventing this is to allow a user to blacklist ports that they don't want Vortex to use, however this requires action by the user and does not prevent mistakes from being made.

The other privacy issue is the aggregation of sensitive information outside of aggregated network traffic. Vortex is specifically designed to require as little information from the user as possible. Currently the only information available to the Vortex backend system is the IP address of the machine as well as a list of in use network ports. If the user wishes, they are able to locally configure Vortex to only use a defined set of ports in case that they don't wish to disclose the port signature of their machine. Depending on the type of backend system in use, further steps can be taken to increase anonymity. For instance, non-interactive backend systems might not need to know the actual IP address of the volunteer machine, in which case Vortex could anonymize the IP and MAC addresses. Previous work has demonstrated anonymization of network data [16,32], and such techniques could easily be implemented in Vortex.

4.4 Security Exposure

Besides avoiding the exposure of private information, CSW systems must not create additional security vulnerabilities on the client machine or their network. The security issues themselves are shared with and have been explored in the context of other aggregation techniques, but the location of the aggregation nodes brings new aspects to the problem. There are two main new aspects to the security problem: exploitability of the wormhole implementation, and risks to the local resources resulting from a compromised backend system. Vortex currently minimizes the former, while leaving the latter as a policy decision for wormhole operators.

The current version of Vortex does minimal processing besides simply encapsulating Ethernet frames and tunneling them to a backend. The only processing performed by Vortex is on the Ethernet level headers themselves, which must first pass through network equipment and a pcap filter that only accepts packets with a valid format. However this might change if Vortex is implemented with additional packet processing capabilities such as anonymization, so such changes must be implemented with great care. Furthermore the packets captured by Vortex are never delivered to the local host, they are simply encapsulated and tunneled through Vortex.

The issue of transmitting traffic from the backend onto the volunteer's local network poses a complicated problem. While this issue is common to honeynets and other traffic aggregators, the fact that the potentially harmful traffic is being inserted onto a volunteer's network leads to a much more sensitive situation. Ultimately this is a policy decision that must be made by the wormhole operators themselves. Even though the current version of Vortex blindly writes all traffic from wormholed ports to the network, the capability to block all or some of the traffic is available through the VTL and VNET frameworks. VNET can be configured to only forward traffic from the wormhole but not to it, and VTL provides packet inspection mechanisms which would allow Vortex to make packet injection decisions based on rulesets run against the packet contents. Other CSW implementations could inject traffic at remote locations seperate from the client's network presence. This would allow full communication between the backend and attacker while not requiring the client to inject any traffic onto it's local network.

5 Invisibility to Attackers

To further evaluate the utility of CSW we investigated the degree to which the wormholes were detectable by an attacker. This section assumes that the CSW wormhole is connected to a honeypot or some other system that emulates an actual service. These systems depend on an attacker believing that their target is actually a legitimate machine, so it is important to understand whether CSW systems provide enough information to tip off an attacker. Furthermore, if an attacker discovers a wormhole then they can simply avoid it, or try to disrupt it [1,3].

The methods an attacker can use to detect the presence of wormholed port fall into two categories: First, because wormhole traffic is tunneled to a remote location, the packet latencies will be larger for wormhole traffic as opposed to traffic handled by the local machine. Second, because a honeypot will be configured differently from a client machine, often with a different OS, packet formats and network behavior will differ between the wormhole and local services. It is beyond the scope of this work to explore the possibility of transforming traffic formats to mimic different hosts, so we only focus on the issue of latency.

For CSWs to be hidden from an attacker, the added latency of the wormhole must fall within the variance of the latency for a local service. This means that the degree to which wormhole latency is masked depends on the connection quality and location of the wormhole host, the backend, and the attacker. Our experiments attempted to capture the different environments under which all three components might operate.

We conducted the experiments by installing Vortex wormholes at various network locations and connecting them to our VM backend located on the Northwestern University network. We then ran latency measurements from PlanetLab nodes located across North America. We measured latency by using tcpdump to time the durations of SYN/SYN-ACK sequences resulting from TCP connection setup requests. We chose to measure the SYN/SYN-ACK sequence because it is handled in kernel and so is independent of application behavior. The Vortex sensors were located on a home network with a DSL Internet connection, a home network with a cable Internet connection, and a Northwestern local area network. For each test we measured the SYN/SYN-ACK latency for a local service and a wormhole service.

The results are given in Figure 9. Each graph is for different client network (DSL, Cable, LAN). In a graph, paired bars compare the local service latency (left bar) with the wormhole latency (right bar). Bar pairs are given for each of the "attacker" PlanetLab sites. It is important to note that each bar represents an average, and standard deviation whiskers are also shown.

As expected the location of the various parties plays a large role in determining the average and standard deviation of the latency of a connection. Neither the DSL nor the cable networks exhibited enough latency variance to effectively mask the presence of a wormhole. Only the wormhole located on the LAN was able to disguise the presence of a wormhole. While somewhat discouraging, the tests do show that the latency is dependent only on the added latency between the wormhole client and the backend system, meaning that the wormhole implementation added minimal latency from packet processing. This suggests that intelligent and dynamic distribution of a backend system over a hosting service such as PlanetLab could help disguise the presence of a wormhole. That is, were the backend itself running on PlanetLab, it could move closer to the client. Furthermore, if the backend were a virtual machine, implementing such movement could be readily accomplished [22].

6 Related Work

Many different communities have sought to harness the unused resources of volunteer machines to perform large calculations or large scale measurements. The most well known of these projects uses donated CPU time to perform extremely large calculations. Projects such as SETI@home [24] and Folding@home [11] have demonstrated considerable success with such an approach, harnessing hundreds

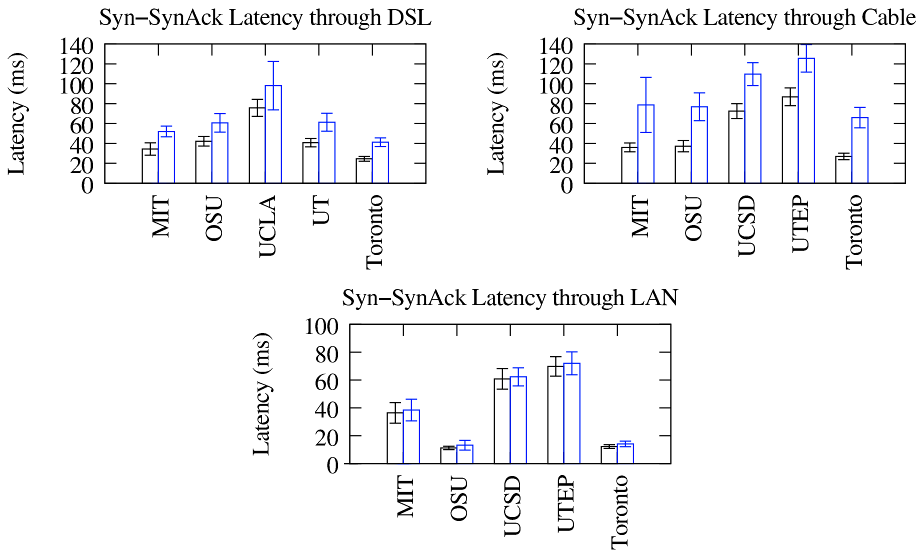


Fig. 9. Differences in latency between a local service and a CSW. Measurements were made by timing the Syn/SynACK sequence caused by the establishment of a TCP connection. Measurements were taken from various PlanetLab sites distributed across North America. For each site the latency of a local service is shown on the left side while the CSW latency is on the right.

of thousands of machines. The Internet measurement community has recently explored such a model, following the realization that widely distributed sensors were necessary to gain a relevant view of the network [8,23]. Measurement and computational clients can all be characterized as *active*, in that they compute or measure something and report the results. CSW clients, however, are *passive*, since they simply tunnel anything they receive back to a backend. While this difference might seem minor, it has serious implications for client privacy and security. Recent work in IDS systems has begun to move towards distributed monitoring as well [2,5,6,19,28], but these systems have yet to demonstrate a technique as readily deployable as a measurement system or computational engine.

To date traffic aggregation techniques have confined themselves to so called *dark address spaces* [14,18,27,33,34]. The idea is to aggregate traffic destined for unused IP addresses and reroute it into a given backend. This usually requires the reconfiguration of network equipment controlling large network domains. While this method of traffic aggregation, commonly referred to as a network telescope, is effective in collecting large amounts of candidate traffic, it has several drawbacks. First, it requires large segments of empty address space. This address space usually can only be found in large organizations such as universities. Accessing this address space requires the cooperation of network administrators for the entire period of aggregation. Furthermore, telescopes are inherently restrictive in the distribution of the aggregated address space. While substantial amounts of traffic can be aggregated from entire dark subnets, such traffic is usually only resulting from automated attacks such as worms or large port scans.

The concept of using wormholes to distributed the network presence of a centralized NIDS backend system was first proposed by Weaver, Paxson, and Staniford [30]. The overall concept is very similar in their work and ours, in that both use distributed wormholes to aggregate traffic into a centralized backend system. However, while their system provides a wormhole for all traffic to a given IP address, we propose to selectively wormhole a subset of traffic based on the network port the traffic arrives at. Additionally, while Weaver, et al propose a hardware solution that requires colocation, the architecture of CSW relies on volunteers donating unused resources of any commodity PC, creating no deployment costs for an operator.

Much work has previously been done in the implementation of actual IDS backend systems, including [7,9,20,29,34]. Our work is aimed at providing traffic aggregation for these systems. Even though most of these systems include their own mechanisms for traffic aggregation we do not propose to replace them, in fact we believe that CSW is a technique that can be used to augment the already present aggregation facilities these systems have in place.

7 Conclusion

In this paper we introduced the concept of *Cooperative Selective Wormholing (CSW)*, a new technique of traffic aggregation for intrusion detection systems. We demonstrated that there is room in the present Internet for CSW systems to achieve adequate address and port coverage, and examined the advantages and disadvantages of CSW compared to present traffic aggregration techniques. We presented a proof-of-concept CSW system, Vortex, and evaluated its performance, including its visibility to volunteers and attackers.

In the future we plan on expanding Vortex to provide a deployable selective wormhole architecture for use by security researchers. We are also looking for opportunities to integrate Vortex into existing honeynet architectures or other IDS analysis systems.

References

1. Allman, M., Barford, P., Krishnamurthy, B., Wang, J.: Tracking the role of adversaries in measuring unwanted traffic. In: The 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet (2006)
2. Bailey, M., Cooke, E., Jahanian, F., Nazario, J., Watson, D.: The internet motion sensor: A distributed blackhole monitoring system. In: Proceedings of the 12th Annual Network and Distributed System Security Symposium (2005)
3. Bethencourt, J., Franklin, J., Vernon, M.: Mapping internet sensors with probe response attacks. In: Proceedings of the 14th USENIX Security Symposium (2005)

4. Claffy, K., Crovella, M., Friedman, T., Shannon, C., Spring, N.: Community-oriented network measurement infrastructure workshop report (2006)
5. Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Zhou, L., Zhang, L., Barham, P.: Vigilante: End-to-end containment of internet worms. In: Proceedings of the twentieth ACM symposium on Operating systems principles, ACM Press, New York (2005)
6. Frincke, D.A., Tobin, D., McConnell, J.C., Marconi, J., Polla, D.: A framework for cooperative intrusion detection. In: Proc. 21st NIST-NCSC National Information Systems Security Conference, pp. 361–373 (1998)
7. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: Proc. Network and Distributed Systems Security Symposium (February 2003)
8. Grizzard, J.B., S Jr., C.R., Krasser, S., Owen, H.L., Riley, G.F.: Flow based observations from neti@home and honeynet data. In: Proceedings of the 2005 IEEE Workshop on Information Assurance and Security, IEEE Computer Society Press, Los Alamitos (2005)
9. Jiang, X., Xu, D.: Collapsar: A vm-based architecture for network attack detention center. In: Proceedings of the 13th USENIX Security Symposium (2004)
10. Lange, J.R., Dinda, P.A.: Transparent network services via a virtual traffic layer for virtual machines. In: Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society Press, Los Alamitos (to appear, 2007)
11. Larson, S.M., Snow, C.D., Shirts, M., Pande, V.S.: Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology. In: Grant, R. (ed.) Computational Genomics, Horizon Press (2002)
12. Libnet, `http://libnet.sourceforge.net/`
13. Libpcap: Libpcap, `http://sourceforge.net/projects/libpcap/`
14. Liston, T.: The labrea tarpit, `http://labrea.sourceforge.net/labrea-info.html`
15. Lundin, E., Jonnson, E.: Privacy vs intrusion detection analysis. In: Proceedings of Recent Advances in Intrusion Detection (1999)
16. Minshall, G.: Tcpdpriv, `http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html`
17. Moore, D., Shannon, C., Voelker, G., Savage, S.: Network telescopes: Technial report. Technical Report CS2004-0795, University of California, San Diego (2004)
18. Moore, D., Voelker, G.M., Savage, S.: Inferring internet Denial-of-Service activity. In: Prcoeedings of the 2001 USENIX Security Symposium (2001)
19. Pouget, F., Dacier, M., Pham, V.H.: Leurre.com: On the advantages of deploying a large scale distributed honeypot platform. In: Proceedings of ECCE'05, E-Crime and Computer Conference (2005)
20. Provos, N.: A virtual honeypot framework. In: Proceedings of the 13th USENIX Security Symposium (2004)
21. Rajab, M.A., Monrose, F., Terzis, A.: On the effectiveness of distributed worm monitoring. In: Proceedings of the 14th USENIX Security Symposium
22. Sapuntzakis, C., Chandra, R., Pfaff, B., Chow, J., Lam, M., Rosenblum, M.: Optimizing the migration of virtual computers. In: Proceedings of the 5th Symposium on Operating Systems Design and Implementation (December 2002)
23. Shavitt, Y., Shir, E.: Dimes: Let the internet measure itself. ACM SIGCOMM Computer Communication Review 35(5) (2005)

24. Sullivan, W.T., Werthimer, D., Bowyer, S., Cobb, J., Gedye, D., Anderson, D.: A new major seti project based on project serendip data and 100,000 personal computers. In: Cosmovici, C., Bowyer, S., Werthimer, D. (eds.) Proceedings of the Fifth International Conference on Bioastronomy. IAU Colloquim, vol. 161, Editrice Compositori, Bologna, Italy (1997)
25. Sundararaj, A.I., Dinda, P.A.: Towards virtual networks for virtual machine grid computing. In: Proceedings of the 3rd USENIX Virtual Machine Research and Technology Symposium (2004)
26. Sundararaj, A.I., Gupta, A., Dinda, P.A.: Increasing application performance in virtual environments through run-time inference and adaptation. In: Proceedings of the 14th IEEE International Symposium on High-Performance Distributed Computing, IEEE Computer Society Press, Los Alamitos (2005)
27. The Honeynet Project, `http://project.honeynet.org`
28. Vigna, G., Kemmerer, R.A., Blix, P.: Designing a web of highly-configurable intrusion detection sensors. In: Lee, W., Mé, L., Wespi, A. (eds.) RAID 2001. LNCS, vol. 2212, Springer, Heidelberg (2001)
29. Vrable, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A.C., Voelker, G., Savage, S.: Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In: Proceedings of the 20th ACM symposium on Operating systems principles, ACM Press, New York (2005)
30. Weaver, N., Paxson, V., Staniford, S.: Wormholes and a honeyfarm: Automatically detecting novel worms. In: DIMACS Large Scale Attacks Workshop (2003)
31. WinPcap, `http://www.winpcap.org/`
32. Xu, J., Fan, J., Ammar, M., Moon, S.: Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In: Proceedings of the 10th IEEE International Conference on Network Protocols, IEEE Computer Society Press, Los Alamitos (2002)
33. Yegneswaran, V., Barford, P., Jha, S.: Global intrusion detection in the domino overlay system. In: Proceedings of Network and Distributed System Security Symposium (2004)
34. Yegneswaran, V., Barford, P., Plonka, D.: On the design and use of internet sinks for network abuse monitoring. In: Jonsson, E., Valdes, A., Almgren, M. (eds.) RAID 2004. LNCS, vol. 3224, Springer, Heidelberg (2004)