Network Performance and Network Intrusion Detection Systems

Jan Seruga
Australian Catholic University
jan.seruga@acu.edu.au

Ha Jin Hwang
Kazakhstan Institute of Management, Economics, and Strategic Research (KIMEP)
hjhwang@kimep.kz

Abstract. The project investigates the potential impact an ethernet bridge based Network Intrusion Detection System (NIDS) would have upon network performance. Three operating systems (OpenBSD, FreeBSD and Linux) were used to test three bridges. The impact of these bridges on network performance while running a NIDS Snort in various modes of data capture were evaluated.

1. Aims of the study

The purpose of this research project is to assess the suitability of an ethernet bridge as a location for a Network Intrusion Detection System (NIDS) sensor [18]. This would enable the sensor to be located at any point on the network solving the problem of location and traffic visibility. One disadvantage of this type of system is the potential impact on network performance.

This project has evaluated the bridging performance of two open source operating systems Linux and FreeBSD. It has also tested the impact upon network performance when these bridges are running a lightweight NIDS Snort [1] and the ability of Snort to detect 'attack' packets whilst under a heavy traffic load.[10]

2. Methodology

Only three operating systems have been identified as capable of supporting bridging. These are Linux, OpenBSD and FreeBSD. Due to the poor performance of OpenBSD in early tests OpenBSD was not tested with a NIDS [15].

The testing software was Netpipe. Netpipe has been selected due the reliability of its testing algorithms, the fact it calculates latency as well as throughput and provides variance values for the throughput figure. For the testing of the Network Intrusion Detection system in alert mode it was desirable to have the NIDS raising alerts, thus placing some load on the system [16]. For this reason Netpipe was modified to send both TCP and UDP packets designed to appear like attack packets. Four different packets were used, two TCP and two UDP. The first TCP packet was designed to imitate a probe of the test.cgi application, a once commonly used application used to test the cgi implementation on a webserver. This TCP packet contained a http packet with the

following request GET /test_cgi?passwd.html. If this packet was delivered to a webserver it is unlikely that it would cause any problems, however it should be detected by a NIDS [14] as it contains the words test_cgi and passwd.

The second TCP packet has both the SYN and FIN flags set. This should never occur in practice as SYN is used to create a TCP connection and FIN is used to end a connection, it is not possible to create and to end a connection with the one packet. Packets with both the SYN and FIN flags set are commonly used to probe systems in order to identify the operating system [2]. Any packet with both the SYN and FIN flags set should cause a NIDS to raise an alert.[17]

The UDP packets are designed to use source and destination ports that should not be used by harmless traffic. The first UDP packet has a source port of 53, this is the port at which a DNS server would normally listen on. However no traffic should be coming from this port. The second packet has a source port of 11011 and a destination port of 555. These ports are commonly used by a number of trojan programs including Phase Zero, iNi Killer and Stealth Spy.

The 'attack' packets are transmitted after each iteration of the throughput packet transmission. As mentioned earlier each throughput calculation is based upon the time taken to transfer packets of the same size for a duration of approximately 0.5 seconds.

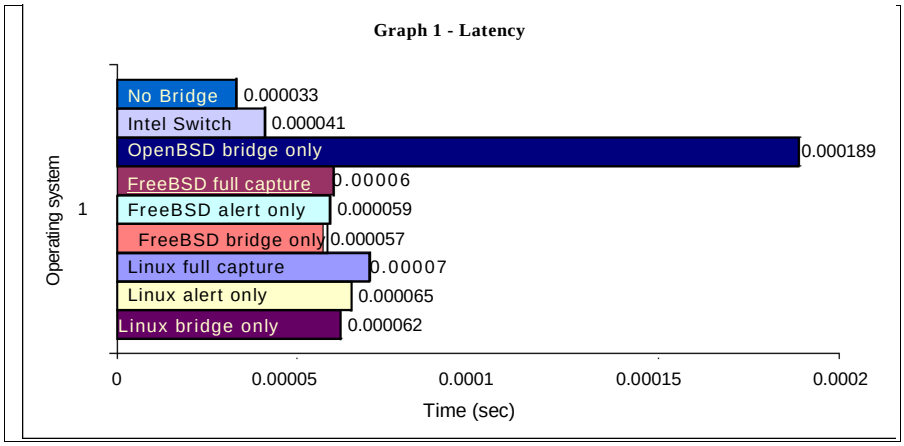


Fig.1. Latency

3. System design

The physical configuration of the testing system conforms to the second design recommended by RFC 2544 [3]. This system requires two testing machines, a sender and receiver either side of the device under test (DUT). By separating the sender and receiver maximum throughput is achieved, and the possibility of either test device influencing the other is reduced. Each device is linked by a cat 5 crossover cable, this allows direct connections between two ethernet devices without using a interconnection device such as

a hub or switch. For baseline tests (where the DUT was not part of the system) a single crossover cable was used to directly link the sender and receiver.

Each operating system was installed on a different hard drive within the same system and was physically connected for a test.

Linux

The Debian 2.2 (potato) distribution of Linux was used for the bridge, however the kernel was upgraded to 2.4.10. [13] Only the most basic applications were installed on the system. Bridging in Linux is controlled by brctl[4]. The bridge was configured in the most basic manner, a bridge br0 was created and both ethernet cards bound to it.

FreeBSD

As with Linux a minimum installation of FreeBSD 4.1.1 was made [12]. A custom kernel was compiled in order to support bridging and remove unneeded drivers[5]. Starting the bridge in FreeBSD simply required a single system call (sysctl -w net.link.ether.bridge=1), which was made manually each time the system was booted.

OpenBSD

Again a custom kernel was compiled for OpenBSD using kernel version 2.9[6]. Bridging in OpenBSD was started automatically with a script. During the testing process it was found that the network devices were failing to auto negotiate their connection and frequently dropped from 100mbps to 10mbps. In order to ensure this connection was maintained at 100mbps both devices were configured to a 100mbps full duplex connection using ifconfig. (ifconfig de0 media 100baseTX mediaopts full-duplex)

Snort [11]

Snort was compiled and installed on both operating systems from the compressed file available for download from www.snort.org. The version used for all tests is 1.8.2. Identical configuration files and signature libraries are used on all systems. Only alert rules are used in the libraries, this means that no logging of packets made, unless the packet matches a known attack signature.

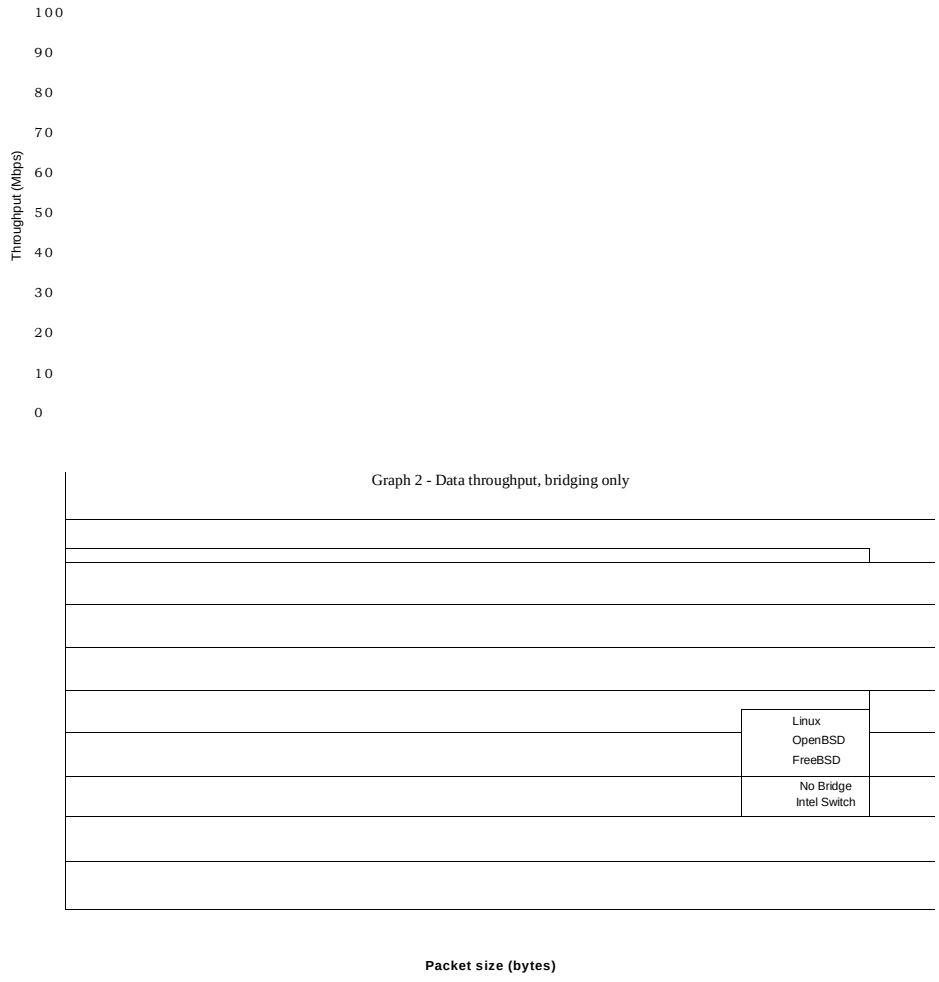
4. Testing process

Two tests were used to evaluate the network performance. The first test was a half-duplex test and the second a full-duplex test. For the half duplex test Netpipe was run as a sender on one system and receiver on the other. As Netpipe transmits a packet to the receiver and waits until it is returned before sending the next one there is only traffic travelling in one direction on the network at any time during the test. In order to ensure both copies of Netpipe start simultaneously, scripts were written to start the test and both tests were scheduled using “at”. Each testing system had it system time synchronised with a separate system using the rdate (remote date) function. This ensured that both applications were started at the same time. Scripts were written to ensure that the configuration of Netpipe was identical for each test.

5. Latency

Graph 1 - Latency of all systems

Latency is calculated by measuring the delay for traffic to be sent from to the sender to the reciever and then returned. As the sender and reciever are in close proximity (4m) the latency times will be low.



Graph 2 - Data throughput, bridging only

It is clear that using the bridges increases latency. The operating system with the lowest impact is FreeBSD, even when performing a full capture the latency is 0.00006s. This is a increase in latency of 82% over a direct connection and a increase of 46% over a specialised networking device. On the other hand the Intel switch increased

latency by 24% when replacing a direct connection.

OpenBSD

The most surprising result was the poor performance of OpenBSD, with a increase in latency of 473%. OpenBSD also performed poorly in the throughput tests when acting as a bridge only.

FreeBSD

One interesting result of these tests is the low impact of running a IDS on FreeBSD[8] [9]. On both of these systems the IDS had a very small impact upon the latency of the system. FreeBSD was the best performer in this area with a increase in latency of 5.3%

between the system bridging only and running a full packet capture. When FreeBSD was running Snort in alert mode only the increase in latency was 3.5%.

Linux

Snort operating on Linux increased latency by 13% in full capture mode and 4.8% in alert mode. This performance was well below that of FreeBSD. Furthermore the impact of Snort on this system was the inverse of FreeBSD. While enabling a full capture on FreeBSD lead to a 1.8% increase in latency compared to running Snort in alert more. On Linux there was a 8.2% increase in latency when Snort was running in full capture mode, compared to running Snort in alert mode.

6. Throughput

The results from the throughput measurements have been represented using four different graphs. Throughput is measured both in terms of number of frames per second or data per second, where data is measured in megabits (1,000,000 bits). These throughput measurements can be graphed relative to packet size (bytes) or test duration (seconds). On the throughput by packet size graphs each point on this graph is distributed evenly (not relatively) along the x axis. The result of this is that the increase in throughput appears to be occurring in a approximately linear manner. The benefit of this graph is that differences between results of different systems are more apparent.

Throughput in terms of data per second is measured and recorded by Netpipe. The frame rate has been calculated based upon the packet size and throughput, using the following formula in microsoft excel.

Frame rate = throughput/packet size * ((packet size / 1500)rounded up to the nearest integer) * 1000000

This formula accounts for a maximum frame payload size of 1500bytes. If the packet size is greater than 1500B then the packet must be divided across multiple frames

7. Impact of bridge on network performance

The results from this research indicate that the level of impact varies depending upon the operating system. Of the operating systems investigated FreeBSD has the least impact and OpenBSD has the greatest impact. There is also a difference between the type of performance impact between different operating systems. FreeBSD has less impact on high frame rates (offered load of 27271.6fps to 23275.3fps) than either Linux or OpenBSD. However Linux has better performance on medium to high levels of data throughput (offered load of 16.77Mbps to 86.04Mbps).

The impact of operating system bridges differs from the impact of the dedicated networking device tested, an Intel InBusiness 8 port switch. The switch was able to support both high frame and data throughput with little impact. The greatest impact on throughput (a reduction in throughput of 35.33%) occurs as packet size approaches maximum frame size, before fragmentation occurs at the network layer. As the level of fragmentation increases (as packet size increases above 1500bytes) the impact of the switch on throughput decreases.

The pattern of impact on throughput caused by the switch differs from the pattern of impact of the operating system bridges. As with the switch all of the bridges tested showed maximum impact upon throughput when packet size was smaller than the maximum size of the ethernet frames (maximum transfer unit (MTU)). However the impact upon throughput of the bridges is roughly linear (between -42.8% to 43.8% for FreeBSD and Linux and between -52% and 60.35% for OpenBSD) until fragmentation begins to occur (ie packet size starts to exceed 1500bytes). Once fragmentation begins the impact of the bridges starts to reduce. At this point the Linux bridge caused a reduction in throughput of 1.65% and the FreeBSD bridge 1.86%. At this point the OpenBSD bridge caused a reduction in throughput of 33.21%. Due to the significant overall impact of the OpenBSD bridge on network performance the decision was made that a OpenBSD bridge would not be a suitable platform for a NIDS sensor.

Graph 3 - Frame throughput, bridging only

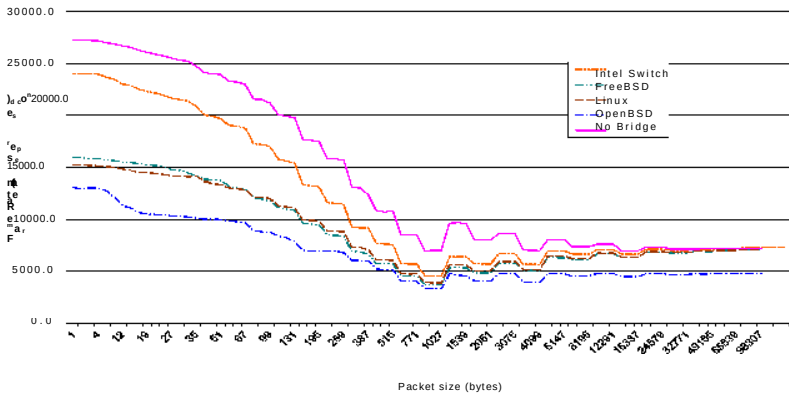


Fig. 2 Frame throughput, bridging only

The cause of the different manner of impact of different bridges upon network performance was beyond the scope of this study. However it would appear from the results that the limiting factor in all systems was the number of frames per second each device could process. At high levels of data throughput the performance of the Linux and FreeBSD bridges approached the performance of the Intel InBusiness switch. It is possible that if higher quality network cards were used a higher frame throughput could be achieved and the impact of these devices upon network performance could be reduced.

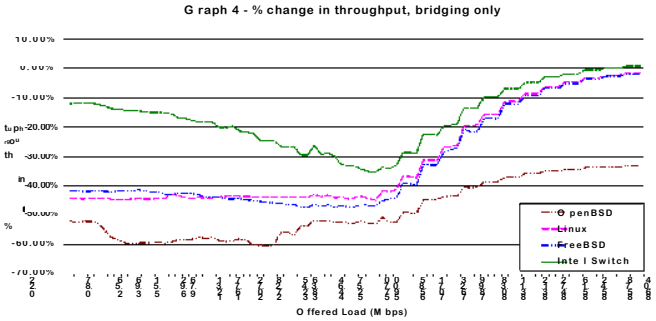
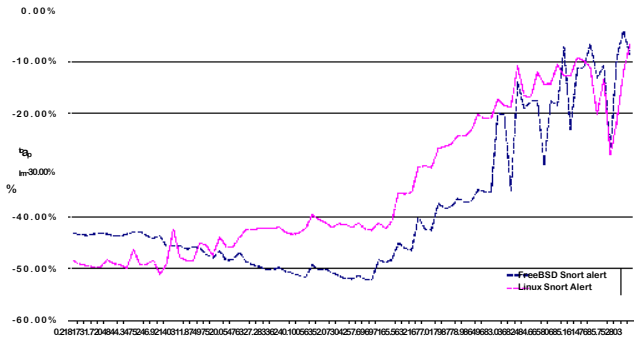


Fig. 3 Change in throughput, bridging only

8. Revelance of results to a production network

Determining the impact these results would have on a 'real life' network is challenging. The offered network load is not consistent with a traffic flow on a production network. In a production network the traffic would be far more bursty, and packets of different sizes would be transmitted in different sequences. The type of traffic would also be determined by the location of the bridge. A bridge placed between a LAN and the internet would experience different loads than a bridge placed between a server and the LAN. Despite this the results from these tests can be used as a indication of the impact on network performance of a bridge based Network Intrusion Detection System. Firstly it is clear that the device would have some impact upon network performance. Latency



Graph 6 - % change in throughput, full duplex traffic, NIDS in alert mode

Offered load (Mbps)

would increase and at low levels of data throughput total throughput would be reduced. The level of impact would be determined by the operating system used for the bridge and the manner in which Snort (or another NIDS) was configured to detect network intrusions.

9. Conclusion

The results from this project indicate that an operating system bridge supporting a Network Intrusion Detection Systems sensor will have a significant impact upon network performance. The level of impact is dependant upon the operating system used, amount of traffic collected by the NIDS sensor and type of traffic forwarded to the bridge.

If the performance of the bridging abilities of either Linux, FreeBSD or OpenBSD can be improved there is a high likelihood that a NIDS sensor could be located on a bridge with minimum impact upon network performance However the current bridging performance of Linux, FreeBSD and OpenBSD is not sufficient to support a NIDS sensor.

.

Reference

1. Haoyu S., Sproull, T., Attig, M., and Lockwood, J. (2005) Snort offloader: a reconfigurable hardware NIDS filter. International Conference on Field Programmable Logic and Applications.
2. Northcutt, S. and Novak J (2002). Network Intrusion Detection.. Indiana, New Riders.
3. Bolla, R. and Bruschi, R (2006) RFC 2544 performance evaluation and internal measurements for a Linux based open router. Workshop on High Performance Switching and Routing.
4. Yan, F and Yeung K (2008) The development of novel switching devices by using embedded microprocessing system running linux. Proceedings of the 2008 IEEE international parallel & distributed processing symposium.
5. Veytser, L. and Cheng, B (2011) An implementation of a Common Virtual Multipoint Interface in Linux. 2011 - MILCOM 2011 Military Communications Conference
6. Liguo, Y., Schach, S., Chen K., Heller, G., and Offutt, J (2006) Maintainability of the kernels of open-source operating systems: A comparison of Linux with FreeBSD, NetBSD, and OpenBSD. Journal of Systems and Software.
7. Crowley, P.,Franklin, M., Hadimioglu, H., Onufryk, P. (2003) Network Process Design. Issues and Practices Volume 3, Elsevier
8. Vasilidis, G., Antonatos, S., and Polychronakis, M., Markatos, E., and Ioannidi, S. (2008) Gnort: High Performance Network Intrusion Detection Using Graphics Processors. Springer Link. Lecture Notes in Computer Science, Volume 5230/2008
9. Chebrolu, S., Abraham., A., Thomas, J. (2005) Feature Deduction and Ensemble Design of Intrusion Detection Systems Computers & Security.Volume 24, Issue 4. Elsevier.
10. García-Teodoro., P., Díaz-Verdejo, J., Maciá-Fernández, G., Vázquez, E (2009) Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & Security. Volume 28, Issues 1–2, Elsevier.
11. Aickelin, U., Twycross, J., and Hesketh-Roberts, T. (2007) Rule generalisation in intrusion detection systems using SNORT. International Journal of Electronic Security and Digital Forensics. Inderscience Publishers.
12. , K., and Kahtani, A. (2010) Performance evaluation comparison of Snort NIDS under Linux and Windows Server. Journal of Network and Computer Applications. Vol 33, Issue 1. Elsevier.
13. Salah, K. (2008) Boosting throughput of Snort NIDS under Linux. International Conference on Innovations in Information Technology.
14. Haslum, K (2008) Real-time intrusion prevention and security analysis of networks using HMMs. Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference.

15. Cheng, F., Roschke, S., and Meinel, C. (2009) Implementing IDS Management on Lock-Keeper. Lecture Notes in Computer Science, 2009, Volume 5451/2009. Springer Link.
16. Marchetti, M., Messori, M. and Colajanni, M. (2009) Peer-to-Peer Architecture for Collaborative Intrusion and Malware Detection on a Large Scale. Lecture Notes in Computer Science, 2009, Volume 5735/2009 SpringerLink.
17. Schaelicke, L. (2005) Characterizing sources and remedies for packet loss in network intrusion detection systems. Proceedings of the IEEE International Workload Characterization Symposium.
18. Chandradeep, K.B. (2009) A Scheme for the Design and Implementation of a Distributed IDS. First International Conference on Networks and Communications, NETCOM '09.