2008

# Traffic Analysis of UDP-based Flows in Ourmon

Jim Binkley
*Portland State University*

Divya Parekh
*Portland State University*

# Traffic Analysis of UDP-based Flows in Ourmon
# Portland State University - Computer Science Technical Report - 0807

*James R. Binkley*
*Computer Science Dept.*
*Portland State University*
*Portland, OR, USA*
jrb@cs.pdx.edu

*Divya Parekh*
*Computer Science Dept.*
*Portland State University*
*Portland, OR, USA*
divyap@cs.pdx.edu

## Abstract

We present a custom UDP flow tuple with an IP address key and a set of simple related statistical attributes. Attributes are used to calculate a per host metric called the *UDP work weight* which roughly measures the amount of network noise caused by a host. The work weight is used to produce a near real-time sorted top N report for UDP host tuples. We also present a derived attribute based on an algorithm called the *UDP guesstimator*. The UDP guesstimator roughly classifies port report hosts into various traffic categories including security threats (DOS/scanning) or P2P hosts based on high UDP work weights and other flow attributes. This algorithm does not use Layer 7 data and only relies on Layer 3 and Layer 4 statistics taken from the UDP flow tuple. Although we believe the algorithm to be fairly effective, we discuss several common sources of false positives including DNS servers, and P2P systems which may sometimes appear to be scanners. We also briefly present an experiment that has attempted to determine P2P applications based on UDP packet size histogram bins.

## 1 Introduction

Security threats based on botnets bringing with them denial-of-service attacks, scans, spam and keylogging are rife [6, 8]. In addition, the relatively recent emergence of a pure p2p-based botnet called stormworm [14] has caused concern amongst security professionals who were used to the more traditional TCP-based IRC botnet control channels. Stormworm uses UDP for its command and control. In addition, many peer-to-peer (P2P) protocols including Bittorrent have switched to using a UDP-based control plane based on the Kademlia protocol [12]. Thus there is a clear trend in both botnet and P2P protocols for using UDP for control. As a result there is also a clear need for UDP-based detection schemes for related security incidents and for detecting P2P-based protocols.

In this paper we discuss recent work in our ourmon system aimed at detecting UDP-only threats including UDP scanning and denial of service attacks, and UDP-based P2P. Portland State University has approximately 26000 users and growing wireless and dormitory populations. Not surprisingly we see a great variety of P2P traffic often aimed at providing entertainment including traditional file-sharing applications and newer video and audio applications like Joost [9] and Skype [15], as well as more traditional security incidents including UDP-based denial of service attacks.

Our paper introduces the *UDP port report* which is part of the ourmon [3, 13] monitoring system. The port report consists of UDP port tuples collected every 30 seconds by a packet sniffing ourmon probe. The port tuple consists of a set of statistical attributes, some of which are derived from more fundamental attributes. Each tuple is indexed by the host IP address. Per tuple we compute a measurement called the *UDP work weight* which we consider to be a measure of noise generated by a host. Typically scanners and systems engaged in DOS attacks produce very high work weights. P2P systems (and DNS servers) produce a high work weight when busy. Ordinary hosts produce lower numbers for UDP-based protocols like Microsoft file sharing and client DNS resolution. In addition we have invented another algorithm called the *UDP guesstimator* which is based on a few of the attributes found in the UDP port tuple including the UDP work weight. The guesstimator helps us classify traffic into simple classes like scanner or p2p. Note that this algorithm is only based on Layer 3 and Layer 4 attributes. We also present a limited experiment based on attempting to determine applications based on UDP packet sizes. The results of the experiment are intriguing but not conclusive. However we feel the experiment was promising and would like to encourage other researchers to perform similar experiments.

In Section 2 we discuss the UDP port tuple, work weight, and guesstimator. In Section 3 we present exper-

imental results and problems associated with the guesstimator algorithm. In Section 4 we present our experiment with application deduction based on packet size. In Section 5 we discuss related work and present our conclusions.

## 2 UDP Flow Tuple

The front-end data collector gathers UDP tuples over its thirty second sampling period. Tuples consist of a set of basic attributes with an IP address as an index. At the end of the sampling period the probe software walks the tuple set and for each tuple computes the UDP work weight, which is a measure of per host noise (discussed in the next section). Thus the UDP work weight is added to the tuple set and is a second order computed attribute. All UDP tuples are sorted by the UDP work weight and the first N tuples (set by configuration) are passed to the back-end for further processing and web-based display.

The set of sorted tuples produced during a sample period are called the *UDP port report*. The UDP flow tuple has the following simplified form including the UDP work weight.

```
(IPSRC, WEIGHT, SENT, RECV,
ICMPERRORS, L3D, L4D, SIZEINFO,
SA/RA, APPFLAGS, PORTSIG}
```

The logical key in this tuple is the IP address *IPSRC*. We will return to the work weight *WEIGHT* in the next section. *SENT* and *RECV* fields are packet counts for UDP packets sent to/from the host in question. *ICMPERRORS* are counts for various kinds of ICMP errors (unreachable in particular) returned to the host IP. *L3D* is a count of unique L3 IP destination addresses in the sample period. Likewise, *L4D* is a count of unique destination UDP ports. Note that L3D and L4D provide useful information about possible P2P behavior for the host in question. *L4S* is a limited count mechanism for unique L4 source ports and basically counts up to 10 as multiple UDP source ports are uncommon in nearly all hosts. (We have never seen more than five UDP sockets open). The *SIZEINFO* field is a a histogram of sent packet sizes in terms of the layer 7 byte payload counts (40, 90, 200, 500, 1000, 1500). The *SA/RA* field is a running average for layer 7 payload size of UDP packets sent by the host (SA), and UDP packets received by the host (RA). The *APPFLAGS* field is a user programmable field based on regular expressions and is used for peeking at Layer 7 content. For example we can use it to determine if a packet matches a regular expression for various kinds of P2P protocols (DHT Bittorrent, Gnutella). We also use it to mark hosts sending DNS packets. The port signature (*PORTSIG*) field is a sample of one to ten destination UDP ports seen during the sampling period and can sometimes be used to determine that a particular host is running a particular protocol. It is sorted in ascending port order and also gives a percent of packets associated with the port in question. It has proven useful in the past for spotting scanning botnet clients that may pick on a small handful of destination ports due to built-in exploits.

### 2.1 UDP work weight

We derive an attribute called the *UDP work weight* from a fairly simple set of fundamental per IP host attributes.

$$ww = (SENT * ICMPERRORS) + RECV$$

For one sample period, we take the count of UDP packets sent by the host and multiply it by the total number of ICMP errors returned to the host. If the ICMP count is 0, we promote it to 1. ICMP errors are for the most part unreachable errors. This gives us a noise measurement. We then add in the count of packets returned. Informally if there are high error counts, the work weight tends to be much higher due to multiplication of errors by packets sent. As a result the position of the error prone host in question relative to hosts with less errors is higher in the UDP port report. On the other hand if a host is simply doing a great deal of work either sending and/or receiving packets, we simply add those numbers together to boost it relative to other less chatty hosts. All things being equal sans errors, hosts doing a lot of work will be higher in the port report compared to hosts doing less work. Fundamentally this may be viewed as a top talker metric where error-prone talkers are given a boost.

Per site work weights are dependent on overall network speed, but the values are relative to each other within a site. PSU has a gigabit network. In theory one might calculate that given a rough maximum count of 1.4 million minimum-sized UDP packets per second at gigabit speeds, with a return of 1.4 million ICMP packets, the highest number would be on the order of 2000 billion (2 trillion and bad news but we have never seen a number over a billion). A single system sending around 1.4 million packets with no error packets returned would of course have a weight of 1.4 million. The daily average for the highest work weight is around 75000.

Extreme work weights (say 100 million or above) typically belong to hosts that are either scanning as fast as possible or are engaged in a DOS attack. Lower work-weights in the range 1000-200000 generally tend to belong to hosts that are engaged in some form of P2P or game activity (Bittorrent, Gnutella, Joost video, Counterstrike). This band typically includes our enterprise DNS servers simply because they do a great deal of work and are subject to a great deal of noise due to failed DNS lookups (this subject needs more investigation). P2P

hosts appear here because some percentage of their sent UDP packets go astray. In the case of P2P hosts this is likely because their set of peers is never completely fresh (some have gone off-line or vanished). Lower numbers are of course typically ordinary clients simply engaged in more mundane activities like ordinary DNS lookups or use of typical UDP-based services like the Microsoft distributed file system. Audio users (skype) typically fall in the lower to lower middle bands as well because they do not send large amounts of UDP packets.

As a result our experience suggests that the UDP work weight tends to roughly divide UDP traffic classes up into three levels, based on high amounts of errors, normal with ”medium” amounts of errors or high packet activity, and a lower threshold with small amounts of packets. In point of fact, the first class with high error state associated with scanners or DOS attacks is of course anomalous. In our network a work weight of 10 million or more will only occur a few times a day (in our case typically due to outside in scans of our entire class B address space) but it is not what one normally sees in the average 30-second port report. DOS attacks are rare but have occurred and given that a gigabit UDP-based host can easily send more than 1 million packets per second, very high work weights indeed can occur under those circumstances.

The average normal highest value is on the order of 75000 or so and is typically associated with either P2P hosts or local DNS servers. Our 10 million ”abnormal” threshold is thus chosen to be safely above the normal highest value for any non-attacking host and is somewhat under the values we see for external hosts scanning our entire network space for ”interesting” UDP ports, although slower scanners may only rack up a work weight of a million or so.

We use this particular work weight threshold (10 million) for automated packet recording in the ourmon system. This has proven useful in a number of DOS attacks as the ourmon probe will simply record the first 1000 or so packets using a dynamically generated Berkeley Packet Filter (BPF) expression based on the IP address extracted from the top UDP work weight tuple. An analyst can use any BPF sniffer (e.g., tcpdump) to playback the packet trace. In addition, the web-based ourmon event log typically will succeed in showing the IP address of a single offender even if the probe itself fails due to overload. This is probably due to the system capturing the rising edge of the attack. As a result this recording feature has proved invaluable on occasions when we have had an internal DOS attack aimed at the outside world due to a compromised host.

We will look at example data results for a simplified UDP port report below in Section 3.

## 2.2 Guesstimator

The guesstimator algorithm is a decision tree based on a few attributes derived from the UDP tuple. The attributes in question do not include layer 7 information as we wish to be able to detect P2P like behavior even in the presence of data payload encryption. For each host address we try and formulate a per sample period guess with the intent of roughly categorizing a host into either a ”scan” or ”p2p” sub-class.

The algorithm in rough form is as follows:

```
guess = "unknown"
if udp work weight >
    SCAN_THRESHOLD_WEIGHT
    guess = "scan"
    if L3D is HIGH and L4D is LOW
        guess = "IP scanner"
    else if L4D is HIGH and L3D LOW
       guess = "port scanner"
else if L3D count and
L4D count > P2P_THRESHOLD
         guess = "p2p"
```

Although the real algorithm is a little more complex, it more or less boils down to the above logic. The explicit intent of the algorithm is to point out some obvious characteristics of the UDP tuple. (In the old days this decision tree might have been called an ”expert system”). The various thresholds are rude heuristics that seem to work well. Note that the logic boils down to first testing to see if the host in question has a high scanner weight, if so we assume it may be a security threat. We then try to characterize it as either a IP destination scanner or an IP port walker. The threshold metric here is 10 million. This value catches external scanners typically walking our IP address space at 100s of packets per second (assuming there are enough ICMP errors). It also catches DOS attacks. It is however set high enough that it is out of the range of most ”normal” university p2p peers and DNS server traffic.

If the host is not a scanner, we test to see if it might have p2p characteristics. This test is simple and the counts we use for unique L3 IP address and L4 destination port counts are set very low (10..20). This value might very well be set in the 100s as that is typical for typical p2p applications like Bittorrent and Gnutella using the DHT protocol. One is more or less defining P2P in UDP terms as simply meaning host X is talking to a remote set of hosts at different ports. The exact threshold value here might range from 10 to 100 but a lower number seems to work reasonably well and is more inclusive for better or for worse than a higher number.

The algorithm is in some sense just a summary of obvious attribute values. It boils down to either many er-

rors with a DOS or scan compared to a p2p guess which has less errors but is characterized by many peer IPs and peer UDP ports. The most important notion of the algorithm is that it inherently prioritizes security concerns over more prosaic everyday affairs (at least at universities). DOS attacks are more important than hosts running p2p applications. We have tested it by looking at packet traces, and by running various popular p2p applications in our lab and looking at the results. Empirically it seems roughly accurate although it could certainly be argued that it is simply pointing out attribute combinations. There are certain problems with this algorithm that we will discuss below in the experience section.

We should also point out that our UDP tuple includes a Layer 7 based regular expression mechanism. It has proven useful as a method of cross verification for non Layer-7 attributes as we have useful regular expressions for the DHT protocol use in bittorent-based applications as well as for DNS messages themselves.

## 3 Experience and Problems

In table 1 we give a small and simplified port report table which shows a set of examples. Our examples here as with the real port report are sorted by work weight. IP addresses in this example are merely an index. This table has an interesting feature that is unfortunately not available in the real world. The final column labeled *truth* shows an accurate classification based on packet knowledge. We have chosen examples that also all have accurate pattern matching. 'b' is a tag that means host packets matched a pattern for a DHT peer query (b for Bittorrent although this is the DHT version of Bittorrent). 'd' is a very strong pattern for DNS payloads. 's' is a pattern for "spim" and matches common UDP-based external scans of our entire IP address space looking for Windows boxes that might allow a 1 packet UDP payload to be turned into a popup (as far as we know there aren't any, but never mind). This is a very common external scan performed several times daily against our network and ironically can be seen as a reference scan.

Results due to the work weight sort fall into the large, middle noise, and negligible noise categories as mentioned before. Examples 1, 2 are over 10 million so we consider them to be "very noisy" and hence they are labeled scan. Example 2 is in reality a real "spim" scan. Example 1 might be deemed a false positive (arguable). Examples 3 and 4 are high in terms of overall numbers in a normal port report (not true here because we have only chosen some relevant tuples), and seem to have many ports and many IP destinations therefore they have been labeled as "p2p". Example 3 is really p2p. Example 4 is a DNS server. Of course in a university network there are many variations on example 3 and not very many examples of 4 (as we have a small set of DNS servers). Example 5 is an ordinary client that has merely used DNS for lookups and is a very typical low noise source.

Now let us look at the more interesting examples in which our decision tree produced a somewhat invalid classification. In example 1, it claims that the host in question is performing a scan. This is a somewhat common event. What is actually going on as verified by looking at packet traces is that the host in question is issuing many DHT *get_peer* requests. We have not been able to reproduce such behavior in a lab setting, but we suspect that the problem is somehow based on a stale peer to peer remote host cache. Hence the local box is trying to find mates, but for some reason cannot find some substantial subset of its peers. We have seen such behavior in the past with Gnutella in particular but at that time, Gnutella was purely TCP-based. The pattern match here provides a useful clue. It is also true that an anti-P2P security policy could truly consider this bad. In a university environment one might on the other hand consider it a false positive. One could argue that this behavior is more or less a scan, however it would be nice to distinguish it from other forms of scans.

Example 4 is in fact a local DNS server. DNS servers have many clients and many remote ports and thus by definition fall into our simple definition of P2P (many ports and many IP peers). However this seems spurious on some grounds. It is perhaps enough to simply know the IP addresses of well-known local DNS servers. Furthermore our L7 pattern is very strong and works very well. It might be enough to simply note that a large percentage of packets are sent to port 53. On the other hand, once again, this is cheating and it might be more satisfying if we had a better heuristic means for telling DNS servers from other hosts.

## 4 Guesstimator Application Recognition

Recently we performed an experiment partially inspired by the work in [5] as we found the idea of looking at packet sizes as a means for identification of application types to be intriguing especially in the UDP arena. We speculated that attention to packet sizes might produce a useful attribute that could help in the identification of UDP-based applications or application classes. This might be because UDP-based socket programming may involve the use of certain packet sizes for certain message classes – particularly node and file lookup messages used in p2p protocols as used in the Kademlia distributed hash table protocol[12]. We focused on popular P2P applications like Emule, Azureus, Limewire, Joost and Skype, as well as with another notorious p2p application called "stormworm" (libpcap traces were provided by security parties elsewhere for our experiment).

Table 1: UDP port report examples

| IP | ww | guess | sent | recv | unreach | L3D | L4D | patterns | ports | truth |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 20114684 | scan | 24301 | 17757 | 827 | 208 | 527 | b | [3811,etc] | p2p |
| 2 | 12773740 | ipscan | 6598 | 12 | 1936 | 600 | 2 | s | [1026,1027] | scan |
| 3 | 49420 | p2p | 1555 | 1215 | 31 | 1637 | 1295 | b | [1533,6881,etc] | p2p |
| 4 | 3321 | p2p | 2430 | 891 | 1 | 703 | 279 | d | [53,etc] | dns |
| 5 | 48 | ? | 24 | 24 | 0 | 10 | 1 | d | [53,etc], | client |

Experimentally we focused here on our UDP tuple sent packet histogram SIZEINFO and on our computed SA.RA send and receive packet size attributes. Again the *SIZEINFO* field is a a histogram of sent packet sizes in terms of the layer 7 byte payload counts (40, 90, 200, 500, 1000, 1500). The *SA/RA* field is a running average for layer 7 payload size of UDP packets sent by the host (SA), and UDP packets received by the host (RA). We installed these applications in our lab, ran them, gathered data from UDP port reports and then analyzed the data.

Our analytical technique for the most part boils down to graphing the data within the various packet size bins and within the SA and RA attributes themselves. Note this makes seven attributes in all, five from the SIZEINFO sub-tuple, and then the SA/RA attributes. Within each attribute we then attempted to extract a lower or upper bound that for the most part had all of the data samples in it. This is really only a manual technique for threshold selection within each of the seven possible attributes. We then produced an if statement based on the thresholds and added it to a set of if statements making up a application guess decision tree based mostly on only these attributes. This application decision tree is programmed as an additional "sub-guess" in our guesstimator algorithm. Barring skype, we only run this algorithm when the main guesstimator guesses that the application is a p2p app, thus it is fair to mostly view the application guesstimator as a sub-tree in the guesstimator's decision tree.

In table 2 we show our test application set along with the protocols we tested. Note that some applications like Limewire actually offer both the old Gnutella protocol and at this point Bittorrent as well. In table 3 we show the threshold values that we use for various applications. Note that some values may overlap. As a result we allowed our algorithm to produce more than one guess if it seemed appropriate (even though this may make no sense). Put another way, we did not make the guesses mutually exclusive. Note that the *size 0* label corresponds to our smallest size bin. We found that the largest size bin was not useful at all hence we actually only used six of the seven size attributes. In general we have never seen a host use full-size MTU UDP packets, although it is always possible. It is fair to say that full-size packets are rare.

Note that barring stormworm, our testing was done only with lab-based applications – thus we were testing known applications (on Windows Vista). At this time we have not done testing based on examining packet traces found "in the wild". However for the most part we found unique patterns when all useful size attributes were taken into consideration. Of all the applications we tested we found that in general our mechanism was only partially successful with skype and in general seemed to work well with the other applications.

Another problem area was discovered when we deployed the algorithm in our real network. We have observed that there are sometimes false positives caused by the stormworm guess. Our pattern matching shows that sometimes single 30-second samples may produce a stormworm guess as a side effect of a host running a p2p protocol (like Bittorrent or Gnutella). We know this is a false positive because the host in question is shown to be running a p2p protocol via L7 pattern matching. Also a real stormworm infected machine would exhibit the stormworm pattern over many consecutive samples over long periods. (And of course might attempt to send voluminous amounts of spam on TCP port 25). The p2p machines only produce a few samples worth of the stormworm guess and were never observed to connect on TCP port 25. As with our previous Bittorrent scanner, once again we see that p2p applications can cause annoying false positives.

We do not claim our results are conclusive in any sense, merely suggestive. There is an inherent test problem with applications which is that although we may run lab tests that seem to identify a particular application, we by no means feel confident about claiming that we know all particular p2p applications in use in our network (a university network). Other applications may have similar results. We also do not feel confident that we know all modes of operation for a given application including network-behavior or even application configuration. Skype in particular is very complex in the area of network behavior and we observed that sometimes it does not act as a peer to peer protocol when using UDP. Or worse, it may decide to use TCP and not use UDP at all. As a result, this work could easily be flawed in ways we

Table 2: P2P applications and Protocols

| Application Name | Protocols |
|---|---|
| eDonkey | eMule |
| Bittorrent | Bittorrent |
| Azureus | Bittorrent |
| Utorrent | Bittorrent |
| Limewire | Gnutella/Bittorrent |
| Joost | Joost |
| Skype | Skype |

Table 3: Threshold Table of P2P applications and Protocols

| Application-Protocol | size 0 | size 1 | size 2 | size 3 | sa | ra |
|---|---|---|---|---|---|---|
| Emule- emule KAD and Servers | 30 to 100 | 0 to 15 | 1 to 40 | 0 to 10 | 25 to 70 | 30 to 85 |
| Emule- emule KAD only | 30 to 90 | 0 to 55 | 0 to 30 | 0 to 5 | 40 to 70 | 0 to 85 |
| Emule- emule Servers only | 30 to 90 | 10 to 30<br>80 to 100 | 0 to 10 | 0<br>70 to 90 | 10 to 30<br>80 to 100 | 10 to 30<br>80 to 100 |
| Bittorrent Azureus | 0 to 60 | 20 to 97 | 0 to 18 | 1 to 20 | 50 to 90 | 60 to 136 |
| Bittorrent utorrent | 0 to 3 | 10 to 90 | 0 to 40 | 10 to 100 | 80 to 200 | 41 to 200 |
| Bittorrent Bittorrent | 0 | 1 to 94 | 1 to 30 | 10 to 80 | 94 to 190 | 60 to 160 |
| Gnutella Limewire | 0 to 30 | 20 to 80 | 0 to 50 | 0 to 50 | 41 to 100<br>130 to 150 | 70 to 190 |
| Joost | 30 to 80 | 1 to 40 | 0 to 35 | 0 to 50 | 20 to 135 | 180 to 210 |
| Stormworm | 99 | 0 | 0 | 0 | 20 to 27 | 0 to 10<br>60 to 240 |

do not understand. It is also reasonable to suggest that our work could be improved upon by having more size bins and using more complex data mining techniques. We could also in future work apply it to the DNS protocol as it might help identity it sans L7 techniques. In the final analysis, we believe packet size to be a potentially useful attribute for UDP data. There there is no guarantee that size buckets can stand alone without the help of other attributes. More work is needed.

## 5 Related Work and Conclusions

There are in general two approaches for detecting P2P applications: 1. based on L7 signatures or 2. based on statistical attribute behavior. The signature-based approach relies on payload contents which can be easily changed or can be hidden by encryption or randomization. The other approach is based on statistical study of network behaviors such as application communication patterns. Constantinou and Mavrommatis classify P2P traffic based on connection direction and number of peers in connected group [7] . Karagiannis et al., identify P2P traffic from connection patterns and the concurrent use of UDP and TCP [10]. In later work, Karagiannis et al., introduce BLINC [11] a general classification mechanism that classifies hosts based on the protocol usage and connection. Our p2p counts are based on BLINC's work and we also strongly agree with them in the basic tenet that flows can be soley focused on IP host addresses plus related statistics. Collins et al., [5] distinguish Bittorrent flows from FTP, HTTP and SMTP flows between pair of hosts. They study packet size, volume of packets, failed connections and bandwidth usage. Our packet size work is closest to their work. We study connection patterns in UDP traffic of peers and our metric for application identification is based on packet sizes. Genevieve Bartlett et al., proposed two new metrics including ratio of incoming-to-outgoing connections and privileged-to-non-privileged ports [1]. Binkley, et al. have presented a somewhat similar effort for TCP that gives a TCP work weight[2, 4]. Of course our work here is entirely UDP-based.

In this paper we present a UDP host-centric tuple with basic and derived attributes including packet counts, unique L3 and L4 destination counts, a size histogram, and a destination port signature. Packet counts are used to derive the *UDP work weight* which gives us a combined estimate of network noise and network usage per host. We sort by this metric to produce a top talker 30-second report UDP report. We then introduce the *UDP guesstimator* algorithm which primarily attempts to classify hosts into noisy security threats or p2p users based

purely on a handful of L3 and L4 attributes derived from the UDP tuple. We also use our size attributes to make an attempt at guessing UDP applications, which although no doubt flawed, appears to be potentially useful.

## References


[1] Genevieve Bartlett, John Heidemann and Christos Papadopoulos. Inherent Behaviors for On-line Detection of Peer-to-Peer File Sharing. *In Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007,* Anchorage, AK, USA, May 2007.

[2] J. Binkley, J. McHugh, and C. Gates, Locality, Network Control, and Anomaly Detection. *Portland State University Technical Report 04-04*, `http://web.cecs.pdx.edu/~jrb/ourmon2.ps`, January 2005.

[3] J. Binkley, and B. Massey, Ourmon and Network Monitoring Performance. *Proceedings of the Spring 2005 USENIX Conference, Freenix track*, Anaheim, April 2005.

[4] J. Binkley, and S. Singh, An Algorithm for Anomaly-based Botnet Detection, *In Proceedings of Usenix Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI '06)*, San Jose, CA, July 2006.

[5] M.Collins and M.Reiter. Finding Peer-To-Peer File Sharing Using Coarse Network Behaviors. *In Proceedings of European Symposium On Research In Computer Security,* Hamburg, Germany, September 2006.

[6] E. Cooke, F. Jahanian, and D. McPherson, The Zombie Roundup: Understanding, Detecting and Disrupting Botnets. *In Proceedings of Usenix Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI '05)*, Cambridge, MA, July 2005.

[7] F.Constantinou and P.Mavrommatis. Identifying Known and Unknown Peer-To-Peer Traffic. *In IEEE International Symposium on Network Computing and Applications (NCA)*, pages 93-102, Cambridge, MA, USA, July 2006.

[8] The Honeynet Project and Research Alliance. Know Your Enemy, Tracking Botnets. `http://honeynet.org/papers/bots`, March 2005.

[9] Joost P2P Video - Wikipedia entry `http://en.wikipedia.org/wiki/Joost`, October 2008.

[10] T. Karagiannis, A. Broido, M. Faloutsos, and KC Claffy. Transport Layer Identification of P2P Traffic. *In Proceedings of the ACM SIGCOMM Workshop on Internet Measurement (IMC)*, pages 121-134, Taormina, Sicily, Italy, October 2004.

[11] T. Karagiannis, A. Broido, M. Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. *In Proceedings of the ACM SIGCOMM Conference*, pages 229-240, Philadelphia, PA, USA, August 2005.

[12] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. *In Proceedings of IPTPS02,* Cambridge, USA, Mar. 2002.

[13] Ourmon web page on Sourceforge. `http://ourmon.sourceforge.net`, October 2008.

[14] P. Porras, H. Saidi, and V. Yegneswaran. A Multi-perspective Analysis of the Storm (Peacomm) Worm SRI Technical Report 10-01. `http://www.cyber-ta.org/pubs/StormWorm`, October 2008.

[15] Skype - Internet Telephony `http://en.wikipedia.org/wiki/Skype`, October 2008.