Locality, Network Control and Anomaly Detection

James R. Binkley
Computer Science,
Portland State University
Portland OR, 97201, USA
jrb@cs.pdx.edu

John McHugh
Cylab, Carnegie Mellon University
Pittsburgh, PA 15313, USA
jmchugh@cert.org

Carrie Gates
Dalhousie University Halifax, NS, CA
gates@cs.dal.ca

December 28, 2004

Abstract

Ourmon is a near real-time network monitoring and anomaly detection system that captures packets using port-mirroring on Ethernet switches. It primarily displays data via web graphics using either RRDTOOL stripcharts or via histograms for top talker style graphs. We have developed a theory that network scanning launched primarily by worm programs including TCP and UDP scanners may be caught by monitoring network control data including TCP control packets (SYNS, FINS, RESETS) and ICMP errors, or by monitoring certain carefully chosen metadata such as the flow count itself. In this paper we concentrate on TCP and present a "flow tuple" focused on TCP control data along with some new metrics and a novel reporting scheme called a *port signature* report. We illustrate our ideas with examples of attacks as shown by the Ourmon system, and relate those examples to our control theory ideas.

1 Introduction

Recently John McHugh and Carrie Gates at CERT [7] have presented a theory of anomaly detection based on locality. Multiscale locality has proven to be a key to understanding a wide variety of physical and other phenomena. Locality of program counter and data references turned out to be the key to the design of effective memory paging systems [2]. In this case, the key locality concept is the "working set," a set of memory pages that if maintained in the physical memory of the computer will allow the program in execution to make progress without excessive page faulting. This work was in response to the observation that on some time sharing computers, page faults occurred so frequently that the CPU was mostly idle, waiting for pages containing the next data or instructions to be referenced to be loaded into memory. This phenomenon, termed thrashing, led to a variety of models of program behavior, the understanding of which allowed efficient implementation of paged memory systems.

The thesis of the earlier paper was that locality principles are a key to distinguishing and understanding "normal" behavior in computer systems that may be subject to attack by outsiders. We feel that an understanding of normal is an important step towards understanding that portion of abnormal behavior that represents the actions of malicious users of the system. Our long term goal is to develop a sufficient understanding of the systems with which we work so that we can identify properties that are necessary parts of certain malicious activities, and, with luck, properties that are sufficient to indicate such activities.

In general, locality is manifest when the behavior of the system can be represented by relatively compact clusters in some dimensions of a multidimensional measurement space. Note that we have not used the terms "benign" and "malicious" as surrogates for normal and abnormal. In this context, abnormal means unusual. In some cases, as we attempt to understand why locality appears to characterize normal behavior, we may be able to make a case that certain classes of malicious behavior are necessarily abnormal in that they will necessarily fail to meet the "normal" clustering criteria. On the other hand, we may not be able to identify such behavior with absolute certainty.

In network terms, one may baseline local network data, especially control data, and consequently observe significant anomalies in a baseline, thus detecting that network attacks are occuring. Furthermore, this principle of locality is something that can be built into network-based anomaly detection systems. Locality is represented in terms of network address clustering, in the temporal behavior of sources both internal and external to the network, and in making careful distinctions between normal amounts of control packets, and anomalous amounts of control packets. Ourmon, the subject of this paper, is another lens through which we can perceive manifestations of locality.

Ourmon [8] is a network monitoring system that is somewhat akin to systems like SNMP RMON [17] in that it attempts to capture network data including top talker graphs of traditional flows, packet counts, and counts of general protocol data including TCP versus UDP traffic, etc. It runs in thirty second sample periods using various filters that in general capture integer counts or top talker tuple lists. Ourmon is divided up architecturally into two programs, a front-end and a back-end. The front-end uses some hardwired and some user-defined filters for capturing data and placing condensed and summarized data in a few small ASCII files. These files are then passed at the sample period time to the back-end which graphs the data on the web or analyzes it in reports. Thus Ourmon is a near real-time system, as the worst case delay in front-end to back-end processing is never more than one minute. In this paper we will mostly focus on ourmon's TCP-oriented anomaly detection capabilities. [1]

Ourmon has two fundamental techniques for displaying data on the web in a graphical fashion:

1. RRDTOOL [10] stripcharts based on integer counters. Ourmon uses RRDTOOL to construct graphs based on various individual integer counters. The filter mechanism here is typically based on a user-programmable technique where multiple Berkeley Packet Filter (BPF) [6] expressions can be grouped in a single RRDTOOL

[1] For more ourmon architectural details, please see: *http://ourmon.cat.pdx.edu/ourmon/info.html*.

graph. This is a major graphical tool in the Ourmon system and is often useful for looking at network-wide information and metadata. For example, we use it to graph the total numbers of TCP control packets in our network. See Figure 1 for an example of an BPF/RRDTOOL graph.

2. Top talker lists of information presented as histograms or reports. Top talker lists typically have some form of tuple associated with them that is either keyed on an individual IP source address or on the traditional IP flow tuple of (IP source, IP destination, IP protocol, L4 source port, L4 destination port). Ourmon has numerous instances of such tuples including conventional flow tuples, and control count tuples such as the SYN tuple discussed below. See Figure 2 for an example.

At PSU, during Fall 2003 we noticed that the total count of ICMP flows, which was not graphed at that time, had increased from 100's of flows to over 100000 flows per sample period (30 seconds). Students had returned from summer break and had brought a mass infection of the Welchia/NACHI worm [15] back to campus. This particular worm includes a ping scanner to scan for additional IP addresses, thus many such worms raised the overall ICMP flow count. It should be noted that this was a case of no particular host standing out by itself in terms of network traffic. What stood out was the increase in network-wide traffic for ICMP.

Of course, this is an excellent example of the principle of locality. We knew that ICMP flows at PSU were typically on the order of 100's per period and should not be 100000 per sample period. We were also alerted to a network-wide anomaly, not just an anomaly for a single host. This particular incident caused us to begin our recent work to steer Ourmon in the direction of becoming an anomaly detection system as well as a general network monitoring system. It also led us to propose two network anomaly system design principles that we believe are of general importance and have found to be profitable in our attempts to capture network intrusions both from a network-wide and per source IP host perspective. The two principles are as follows:

1. In anomaly detection, it is useful to focus on network control data. Intuitively there should be fewer control packets than data packets, and errors are significant. For TCP this means SYNS, FINS, and RESETS as well as ICMP errors. For UDP this principally means ICMP port unreachables although other ICMP errors are useful as well. As an example, we have a top talker mechanism that captures TCP SYN attacks and a BPF graph that shows network-wide counts of TCP SYNS, FINS, and RESETS. In general, we can use our BPF/RRDTOOL mechanism to graph network-wide behavior, and use various top tuples to look at the behavior of individual IP sources.

2. Carefully chosen metadata may also be of use for anomaly detection. For example, we originally chose to show the largest top talker IP flows as this is a very conventional way to display flow information. As it turns out, we should also have been graphing the number of flows (as seen below in figure 3). In the previous category we mentioned top talker SYN counting for individual IP sources. We also developed an additional metadata grapher that simply counts the

total number of suspicious systems sending many TCP SYNS, where we arbitrary declare that a certain number of SYNS - FINS per count period is "interesting". As a result, we have observed what appear to be coordinated SYN scanning sweeps that originate from many IP sources at a time. This is shown in figure 4.

We should point out that Ourmon at PSU is deployed in our DMZ. Our university has around 26000 students. We use Ourmon to look at our external Internet traffic for a network with over 5000 hosts on it, 300+ Ethernet switches, and 10 routers plus external connections to Internet1 and Internet2. All the graphs presented here come from our central DMZ monitoring station.

Our paper is organized around recently developed anomaly detection mechanisms based on the general locality theory. We illustrate these mechanisms with graphs based on various attacks. We neglect UDP and ICMP in favor of TCP due to space limitations. In section 2 we will look at TCP-based mechanisms that illustrate our control theory notion. In section 3 we look at some metadata examples including our worm counting mechanism. In section 4 we look at validation work coupled with the port signature report. In the last sections we present related work, and our conclusions.

2 TCP Anomaly Detection

In this section, we are going to look at two filter mechanisms that are both focused on TCP anomaly detection. We begin with Figure 1, where we show a daily (last 30 hours) picture using three BPF expressions in a RRDTOOL stripchart. This picture shows the total TCP SYN, FIN, and RESET packet counts for PSU traffic to and from the Internet for a period slightly over thirty hours. As usual with RRDTOOL graphs, "now" (10:00 AM) is on the right hand side. The stripchart moves to the left every thirty seconds. The top curve is the number of SYNs and the size of this curve has basically suppressed the FIN and RESET count lines, barring a small 6:00 am spike in the RESET line. Clearly the spikes in the SYN line indicate one set of major anomalies.

These attacks were caused by large-scale SYN attacks coming from the Internet into PSU during the time period.[2] At the bottom the graph shows that the variation in average SYNs to maximum SYNS was about 1 to 3. In other words a single attack could nearly triple the number of incoming SYNS. From experience we know that PSU's overall traffic is typically diurnal with peaks in the early afternoon. This graph makes a strong suggestion that the number of SYNS during the entire period at PSU is too high, and that the network is seeing a fair number of TCP SYN scans. One might expect that the number of SYNS and FINS would somehow march together even if there were less FINS. Of course, long-term baselining can help resolve this issue, but as this graph is new no such baseline currently exists. We should also point out that the distributed attack here is the same as shown in Figure 3 and Figure 4, which we will discuss later in our metadata section.

Our second TCP filter is based on a new top talker tuple, which is loosely modeled on our original flow list 5-tuple as found in Cisco's netflow tool [1]. We call this the *SYN list* tuple. It has a number of outputs or views in the back-

[2] We believe these attacks are distributed and coordinated SYN attacks that are looking for exploitable Microsoft systems.

end including a histogram sorted by the top IP source senders of SYNS (see Figure 2), the worm counter, and the port signature report, covered in section refvalidation. The tuple stored by the *SYN list* has the following rough form:

```
(IP source address, SYNS, SYNACKS,
FINSSENT, FINSBACK, RESETS,
ICMP ERRORS, PKTSSENT, PKTSBACK,
port signature data)
```

The logical key in this tuple is an IP source address. SYNS, FINS, and RESETS are counts of TCP control packets. SYNS are counts of SYN packets sent from the IP source, and SYNACKS are a subset of only those SYNS sent with the ACK flag set. FINS sent both ways are counted. RESETS are counted when sent back to the IP source. ICMP ERRORS refers to certain ICMP errors like unreachable or TTL errors returned by receivers. The PKTSSENT counts the total packets sent by the IP source, and PKTSBACK counts the total pkts returned to the IP source. There is also a small fixed set of sampled TCP destination ports that we will discuss more below in section refvalidation. This tuple captures the idea of two-way data exchange in a number of ways including counters PKTSSENT and PKTSBACK, FINS BACK, RESETS, etc.

There are currently two weights associated with the SYN tuple, which we call the *work weight*, and the *worm weight*. The *work weight* is computed per IP source as follows:

$$(S_s + F_s + R_r)/T_{sr}$$

and is expressed as a percent. The rough idea here is that we take control packets likely to be used in an anomalous way and divide that count by the total number of TCP packets. Obviousally 100% here is a bad sign and implies a true anomaly of some sort. Such a value is typically associated with a scanner or worm. On the other hand, if ordinary data packet exchange has occured, the weight will be lower and very well may be 0% for classic long-lived connections like FTP. FINS sent are included to capture FIN only scans. Resets returned are included because they are generated by attacks and we want an attack with SYNS producing RESETS to tend to 100%. We will discuss the *worm weight* in subsequent sections.

In the top SYN graph (Figure 2) our histogram labels show the IP address, followed by the FIN (f), RESET (r), total count (t), and *work weight* plus an additional "worm" flag. The worm flag is based on the *work weight* metric and is set to "W" if that metric is 90% or more. In the graph the line below the FIN and RESET counts shows the SYN count used for sorting.

A few packets of (simplified) tcpdump output for the top host in the graph are as follows:

```
131.252.X.Y.3885 > 10.0.0.1.445: S
131.252.X.Y.3886 > 10.0.0.2.445: S
131.252.X.Y.3886 > 10.0.0.3.445: S
```

In other words, the PSU host in question is performing a port 445 (Microsoft file share) TCP SYN scan of external IP hosts. It has a virus and is searching for other hosts to infect.

Our first generation SYN tuple merely counted SYNS, FINS, and RESETS, and sorted on SYNS. We quickly learned that in general many SYNS and no FINS was a sign of an infected host. However we also found that we had false positives in the sense that a few IP hosts would commonly register work weights in a range less than 70% but higher than say 10%. Your average garden-variety worm would have a higher value, typically 100%, but it was not clear at first the reason for intermediate range values. We also performed a modest statistical analysis of multiple million packet samples gathered both during "normal" and "abnormal" times where abnormal meant large external SYN attacks on PSU were underway (as seen by our tworm graph in Figure 4). This showed that in general hosts tender to cluster around low weights, or high weights (during attack periods), but medium weights (between 30..70%) were much less common.

Eventually we determined via manual means (tcpdump for ports and ngrep for content) that the majority of hosts in the middle range were running P2P clients of various forms. P2P systems may generate high rates of SYNS with less successful numbers of connections with peers, hence they may have non-zero work weights. However in general they are doing some work, hence medium (and low) range work weights are common. Researchers in general should be aware that P2P clients may cause false positives if one simply counts SYNS – although we found that in general with our particular metric more modern P2P applications like BitTorrent (compared to Gnutella) have lower but non-zero work weights (say 20% or less). Some apps like Gnutella have higher work weights (say 30% as an average, although higher is possible).

The SYN list graph can at times be paired with the BPF TCP control graph. The BPF control graph presents a network-based point of view and the SYN list gives individual IP sources generating high rates of SYNs. We have seen examples where a spike in the control graph can be matched up to a log entry in the SYN list at the same time. It is also often the case that many IP sources are generating small numbers of SYNS at the same time, thus raising the SYN line on the BPF graph, but not showing any obvious evidence in the SYN list.

The SYN list graph ultimately has been found to be frustrating simply because it did not provide enough data about any individual host. It would indicate a host was interesting, but one still had to resort to a sniffer for details. Also it did not provide any help in information correlation during large scale coordinated attacks. As a result we enhanced the earlier form of SYN tuple to include a small set of destination ports and packet counts for those destination ports, thus laying the groundwork for our port signature report, which we will return to below.

3 Metadata Examples

In this section we look at two useful anomaly detection mechanisms that are "metadata" by which we mean second-order graphs derived computationally from existing filters. Both figures in this section show the same distributed TCP SYN attacks coming into PSU from multiple IP sources over the same time period. These attacks may have used the agobot (or phatbot) tool and involved remotely controlled IRC chat servers [14].

Figure 3 which we will call the *flow count* graph shows the count of flows for IP (all IP flows), TCP, UDP, and ICMP flows respectively. We assert that it is reasonable to view the count of flows in a network as part of the control plane of the local network. Typically top talker flows would show the top flows in terms of bit rate. Here we instead show the total count of all the flows during the sample period and represent the count for the four kinds of flows in an RRDTOOL graph.

In this case, we can see a number of spikes in the count of TCP flows. In general, baselined

RRDTOOL data has shown that PSU traffic in terms of the TCP flow count is diurnal, with perhaps 1k flows at night and 2k flows during the day. Here we see one spike at 9:30 PM on the previous day that has doubled the number of flows to around 4k. Of course in this case we have "flows" of one packet as PSU's IP destination address space (a class B) is being walked by multiple external IP sources. This graph shows a total of 5 TCP spikes, and also one UDP event as well.

Figure 4 which we will call the *worm count* graph is probably our most visually interesting artifact. It is a count of suspicious TCP SYN scanners that is produced as a side effect from the previously mentioned top talker *SYN list*. In this case our front-end takes the entire sorted list, and produces a subset of "interesting" SYN tuples primarily including hosts that satisfy the following *worm weight*:

$$S_s - F_r > C$$

We simply subtract the FINS returned from the SYNS sent and only store the tuple if C, a constant, is greater than some manually configurable constant (which is configurable and defaults to 20 in the current system). We justify C in an intuitive sense by pointing out that in general C should be chosen as large enough that any given IP source in the set is generating more SYNS than FINS, thus one can claim any IP source satisfying C is in some sense "noisy".

Thus the tworm graph consists of IP sources that have satisifed the worm weight. There are three counters shown in the tworm graph, 1. total IP worm weight sources, 2. "us" (meaning hosts that belong to the home network, PSU), and 3. "them" (hosts that are external to the home network). The resulting graph shows a number of sustained attacking periods with IP source counts in the worst case around 1k[3]. We were incredulous at first as to whether or not this filter worked, but the curves produced by it have been verified by hand using tcpdump, and by other graphs including the flow count filter pictured below and the RRDTOOL/BPF graph of TCP control packets (Figure 1). We believe it is producing credible results.

Our subtractive metric seems to work simply because when large attacks are not taking place we are ignoring large numbers of small producers of SYNS and FINS (most applications) and hence establish a stable and small count baseline that may consist of some P2P apps and some single host infections. When an attack occurs, typically most of the attacks produce SYNS with few or no FINS, and hence raise the curve.

The exact value of the constant C is debateable and may be network dependent. Our network is open and has many P2P users and a higher value may be more appropriate. A more conservative network might want to have a smaller less noisy value.

4 Validation and TCP Port Signatures

In this section we wish to discuss our validation and testing of the TCP-based work mentioned previously. We do this in the context of a novel reporting technique called a *port signature* report. This report consists of that subset of the SYN tuple set that barring a few exceptions satisfies the worm weight. It includes metrics like the work weight and other metrics as well includ-

[3] Agobot is capable of spoofing IP source addresses and thus there really isn't any known way at this time to tell how many true IP hosts were involved in these attacks.

ing a small set of 1 to 10 port tuples which provide information about TCP destination ports. Thus this gives us a limited view of ports used by the IP source during the sample period. It also presents information in such a way that attacks from multiple IP sources can be correlated. In this section we will discuss the port signature in more detail using a tabular representation of the port report, and also discuss our validation efforts in terms of both the summarized port report and a small Microsoft-oriented application study we performed to help us better understand what we were seeing in the port report.

Before we present the port signature and related efforts aimed at validation, consider the following statement from Jung, *et. al.*[4]: "Consequently our argument is nearly circular: we show that there are properties we can plausibly use to distinguish likely scanners from non-scanners in the remainder hosts, and we then incorporate those as part of a (clearly imperfect) ground truth against which we test an algorithm we develop that detects the same distinguishing properties". Ultimately in a very narrow sense, it is important to remember, our work weight system catches *anomalies*, as for example in the limited scope of a system sending SYNS and not getting any packets back (barring resets). 100% indicates a true anomaly, but ultimately we do not know if said system is a scanner, a misbehaving program, or a misbehaving worm! We cannot see intent – we can only see the symptoms.

Looking at SYN counts alone does not help much – the next step is to run a sniffer and see if anything can be learned from the packets themselves. This is time consuming and not helpful to IT people who lack time. Further a sniffer trace is not a good way to diagnose an attack in parallel either. We want something that gives us more details and yet at the same time gives us the big picture as a parallel view of multiple IP sources. Looking at a sample set of ports does help us determine in some cases that a particular pattern is an attack, especially when we base our observations in either shared IT experience (via the Internet, local communication between local security gurus and IT staff, the PSU abuse list, and experience gained from cleaning up local infected systems) or as possible, conducting a more formal study of certain applications and their behavior as seen with Ourmon behind a firewall or in a lab. In consequence in some cases, but not all, we "know" we have an attack.

The (somewhat simplified) port signature report given in table 1 consists of a small set of interesting examples taken from one real PSU report from fall 2004. Note that the table is not focused on only high work weights – it is simply one data item out of a composite set. [4]

Each port signature begins with the IP source in question, with statistics for each individual IP source given per line. In addition to three metrics, flags, work, and SA/S, the primary mechanism here is the port signature on the far right of each IP source. The port signature includes 1 to 10 two-tuple port samples, with each port sample consisting of a destination port and a packet frequency count for each port in the port sample space. The number of buckets for port destinations is currently set to 10 (we use ellipsis in the table for cases where the entire port sample space is filled). For example the third entry shows that packets were sent to TCP ports 5554, and 9898 by IP source 3. The former port received 65

The port signature report is sorted in ascending order from top to bottom in terms of its logical key, the IP source address. (Here we are

[4]The real port report can be seen at any time at: *http://ourmon.cat.pdx.edu/ourmon/portreport.txt*.

Table 1: Port Signature Report

ip src	flags	work	SA/S	port signature
1	(WOM)	100	0	[445,100]
2	(WOM)	100	0	[24910,100]
3	(WOR)	100	0	[5554,65][9898,34]
3.1	(WOR)	100	0	[5554,65][9898,34]
4	()	6	100	[1151,1][1905,20] [...
5	()	22	0	[1433,99][3536,0]
6	()	2	10	[1124,14]...[6881,36][6882,5]...
7	(WOR)	100	0	[139,33][1025,22][2745,21][6129,23]

replacing real IP addresses with logical numbers as substitutes – IP source 1 will be referred to as example 1, etc.) A sorted IP source space is useful because one can see "nearby" or same IP source network groupings during distributed IP attacks, and of course, one can easily view ones own IP source address space for outbound attacks. For example, we have observed agobot-based attacks in which all spoofed IP source addresses in a /24 subnet space are attacking the same remote set of ports. In our report above, there are two attacks that appear similar based on their ports coming from the same network (3 and 3.1). The port signature is also sorted from low port to high port and this helps us see similar attacks using the same set of ports. Again the same two "anomalies" (from 3 and 3.1) have the same port signature and are likely to be the same attack.

The flags metric shows us whether or not the worm candidate is receiving two-way data. Flags here include:

1. W - the work weight is 90% or higher.
2. 0 - few fins if any are returned.
3. R - large numbers of resets are being returned.
4. M - few non-reset data packets are being returned.

SA/S is a simple metric that measures the number of SYN+ACK packets sent, which typically are the second packet in the TCP three-way handshake, divided by the total number of SYNS. 0% suggests a client, 100% suggests a server, and some number in between suggests possible P2P activity. This metric does not stand by itself but it is very useful when coupled with other indicators. For example if both the work weight and SA/S are 100%, one is likely seeing a SYN+ACK scan.

The work metric is shown next. In the report, we might choose to show only those IP sources with high work weights (say 80% to be conservative) because of the high rate of "worminess" observed with the work weight. Our IT experience suggests that out of 1000s of instances of such anomalies, we have seen less than 10 cases that were not attacks. These cases are true anomalies in that something is wrong, but they are not necessarily worms. Three example anomalies so far spotted (and explained) include: 1. one case of a popular meeting application that enthusiatically tries to reconnect to its server when the server is taken down for backup, 2. well-known (as opposed to infected) campus email servers that are attempting to forward error messages to spammers (which given fake return IP addresses will never work), and 3. certain P2P clients (often Gnutella-based) that have a very low success rate for peer connections. As a result at this point in time, we are satified with our general understanding of the work metric in the high range. However there are a number of reasons to look at lower work weights for hosts that have satisifed the worm metric. One of the more fundamental reasons is that the port report itself may capture some attacks (as in example 5) where the work weight is low. In such examples, other criteria such as ports are useful.

In an attempt to characterize a rather important set of Microsoft ports that show up over and over in our port report, we captured packet traces of typically a million packets apiece from clients and servers protected by a firewall from the Internet. Here we focused on specific Microsoft ports used by the Microsoft file share system (ports 135-139 and 445) and Microsoft SQL server applications (TCP port 1433). We dumped their associated SYN tuples during short and long sample periods to see if these ports would show up in the worm metric sample or the hosts would have high work weights. The answer was no, which conformed to the intituion of various local security experts. This is not surprising given that TCP connections with these applications are typically long-term. This gives us confidence that TCP ports in these ranges that appear in the port report are likely attacks.

Now let us look at the examples chosen as representative of certain classes of phenomenon. Examples 1, 2, 3, 3.1 and 7 show work metrics at 1003.1 are examples of the dabber worm[5]. Example 7 is an old phenomenon seen many times, and is some form of phatbot/agobot attack. These two examples taken together illustrate a very interesting forensic possibility which is that the display of the ports in some cases (not all) may allow you to identify the worm. On the other hand, Example 2 is a new phenomemon as of late November, 2004 which we have not seen before but based on experience and the work metric, it is highly dubious. Still we have not as of yet identified it.

At a lower weight, example 4 shows something we call the noisy web server phenomenon. This always appears as an external IP address. Certain web servers seem to exhibit this behavior possibly because of large numbers of small TCP connections due to active web page displays. The work weight tends to be low, thus there is two-way data exchange. The SA/S metric is useful here and suggfests these systems are indeed servers. This is apparently a benign phenomemon but our understanding of this behavior could stand improvement.

Example 5 has a low work weight, and yet we know from the previously mentioned application testing that any mention of port 1433 in the work report (with large numbers of packets) is an attack. The work weight is low here because this is a password guessing attack on SQL servers, thus there is (nefarious) work being done. Here the use of ports is invaluable. It is also important to remember that one does not need a high work weight to have an attack.

Example 6 is interesting, as it is quite common to see P2P applications appear in the port signature report. Again this is because peering P2P hosts will have some subset of unavailable peers. Of course there is no guarantee that a given P2P application is bound to a given port. Still we surmise that this example is using BitTorrent because of ports 6881 and 6882. The

SA/S metric is interesting here in that it suggests the host in this example has some server tendencies although it tends to the client side. In general, we intend to do more research on the lower work weights. For example, we hope to improve our abilities to identify P2P applications.

5 Related Work

Until quite recently, scan detection has received relatively little attention compared to other intrusive activities. Part of the reason is the ubiquitous nature of scan and scan-like background data. Another has been the relatively primitive measures used in many intrusion detection systems for dealing with scans. The typical IDS detects scans with a relatively simple threshold measure. SNORT[12] is typical of this approach.

Bro[9] is similar but somewhat more stateful. Recent work by Stolfo's group at Columbia[11] can detect much slower scans as well as some distributed scans by associating scan state with a subnet address rather than an individual IP address. In addition, he has established a network of detectors that exchange information about scans detected at widely separated locations. Work at Silicon Defense[13] collects statistically anomalous events over a long time period and attempts to cluster them into distinct surveillance attempts. We note that this is another example of locality applied to scan detection. The cited paper also contains extensive background information on the surveillance detection problem.

More recently, Jung, *et. al.*[4] have looked at the ratio of connection attempts to connection successes as a function of network occupancy to choose between competing hypotheses that the source of the attempts is a scanner who does not know the network structure or a benign user who occasionally fails to make a connection due to broken URL links or faulty DNS information. One principle difference between Ourmon and this work it that Ourmon also includes the port signature report which both helps determine the nature of the attack and shows locality information about attacks in parallel. Also perhaps the nature of the networks in question may influence the results. Perhaps PSU sees more P2P traffic due to the large local population of students?

6 Conclusion

In conclusion, we suggest that network control data may be viewed as a rich source of information about normal network locality. In particular, anomaly detection may make use of network control data as represented by TCP control packets, flow counts, ICMP errors, and in the case of individual IP sources, counts of packets sent to and returned from the IP sources themselves. We have shown graphs and reports that exploit this phenomenon which are based on either using simple integer counts or top talker graphs (histograms). In general, RRDTOOL graphs give us a network point of view. The histograms and port signature report give us a per IP source view.

Our control theory notion has suggested a number of interesting new tuples and metrics including:

1. A per IP source TCP SYN tuple that includes two-way data. This SYN tuple can be used for multiple outputs including a port signature report which shows anomalies in parallel, and a top talker histogram graph that shows sources sending the largest numbers of SYN packets.

2. We have derived two metrics from this SYN tuple including a *work weight* metric that gives us a simple way to determine if an IP source is simply bombarding us with packets or if there is genuine two-way work going on between that source and remote destinations. Our *worm weight* metric gives us a set of IP sources that seem to be "noisy" in terms of sending more SYNS than FINS, and when combined with the work weight and other data helps us find scanning anomalies.

3. Our port signature report gives us insight into parallel anomalous sources. This can reduce the time spent trying to characterize the activity of a remote set of sources, which may be exhibiting a known anomaly or running a P2P application.

4. For reasons of brevity in this paper, we have neglected discussion of UDP/ICMP scanners. However we have also developed a UDP work weight based on two-way exchange of UDP data, and returned ICMP errors.

One important thread in these tuples and weights is the notion of two-way data. Data returned may either lend credibility to the notion that real work is going on, or detract from that notion if the packets returned are errors. Another important aspect of these tuples are error counts. For TCP that primarily means RESETS and for UDP, ICMP errors.

Although we feel our work reported here is exciting, it is also recent and preliminary. Anomaly detection work takes time and must be based on long-term analysis and long-term baselining of normal (and abnormal) data. As an example of even a simple metric that needs study, consider our RRDTOOL/BPF graph of SYNS, FINS, and RESETS in the PSU network. We are not sure what ratio of SYNS to FINS is reasonable in the PSU network? One might also ask what the ratio should be for particular kinds of applications (email or web), particular hosts, subnets, autonomous systems, and the Internet as a whole? In other words, what kinds of "localities" might exist, and what might one expect a healthy locality to look like? Such information would be invaluable for determining the health of that locality.

In the near future, we hope to take our various counters and metrics per IP source including SA/S counts, two-way flags, the work weight, and the port signature tuples and determine if we can use these techniques possibly coupled with Bayesian statistical methods to detect P2P flows or attacks in a more general way. We intend to study the worm weight as well and analyze the constant used there to determine under what circumstances a different constant might be useful.

7 Acknowledgements

Thanks to Cory Bell, Suresh Singh and Bart Massey for their suggestions and criticism.

References

[1] Cisco Systems. Cisco CNS NetFlow Collection Engine. `http://www.cisco.com/en/US/products/sw/netmgtsw/ps1964/products_user_guide_chapter09186a00801ed569.html`, April 2004.

[2] E. G. Coffman and P. J. Denning. *Operating Systems Theory.* Prentice-Hall Inc., 1973.

[3] C. Gates, J. McHugh, and J. Binkley. An analysis of threshold random walk. 2004, Submitted to RAID 2004.

[4] J. Jung, V. Paxson, A. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In Proceedings of the IEEE Security and Privacy Conference, Oakland, California, May 2004.

[5] Lurhq Corporation virus information. `http://www.lurhq.com/dabber.html`, Dec 05, 2004.

[6] S. McCanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-level Packet Capture. *Proceedings of the Winter 1993 USENIX Conference*, San Diego, January 1993.

[7] J. McHugh and C. Gates. Locality: A new paradigm for thinking about normal behavior and outsider threat. In *New Security Paradigms Workshop*, Anscona, Switzerland, August 2003.

[8] Ourmon web page. `http://ourmon.cat.pdx.edu/ourmon`, May 2004.

[9] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In Proceedings of the 7th USENIX Security Symposium, San Antonio, Texas, January 1998.

[10] RRDTOOL web page. `http://people.ee.ethz.ch/~oetiker/webtools/rrdtool`. November 2003.

[11] S. Robertson, E. Siegel, M. Miller, and S.Stolfo. Surveillance Detection in High Bandwidth Environments. In Proceedings of the 2003 DARPA DISCEX III Conference. April, 2003.

[12] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In Proceedings of the USENIX LISA '99 Conference, Novemember 1999.

[13] S. Staniford, J. A. Hoagland, J. M. McAlerney, "Practical Automated Detection of Stealthy Portscans," Seventh ACM Conference on Computer and Communications Security, Athens, Greece, 2000.

[14] Symantec virus information, W32.HLLW.Gaobot.gen. `http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.gaobot.gen.html`, May 05, 2004.

[15] Symantec Virus Information, W32.Welchia.Worm. `http://securityresponse.symantec.com/avcenter/venc/data/w32.welchia.worm.html`, August 18, 2003.

[16] Tcpdump/libpcap home page. `http://www.tcpdump.org`, September 2003.

[17] Waldbusser, S. Remote Network Monitoring Management Information Base Version 2. IETF. RFC 2021, January 1997.

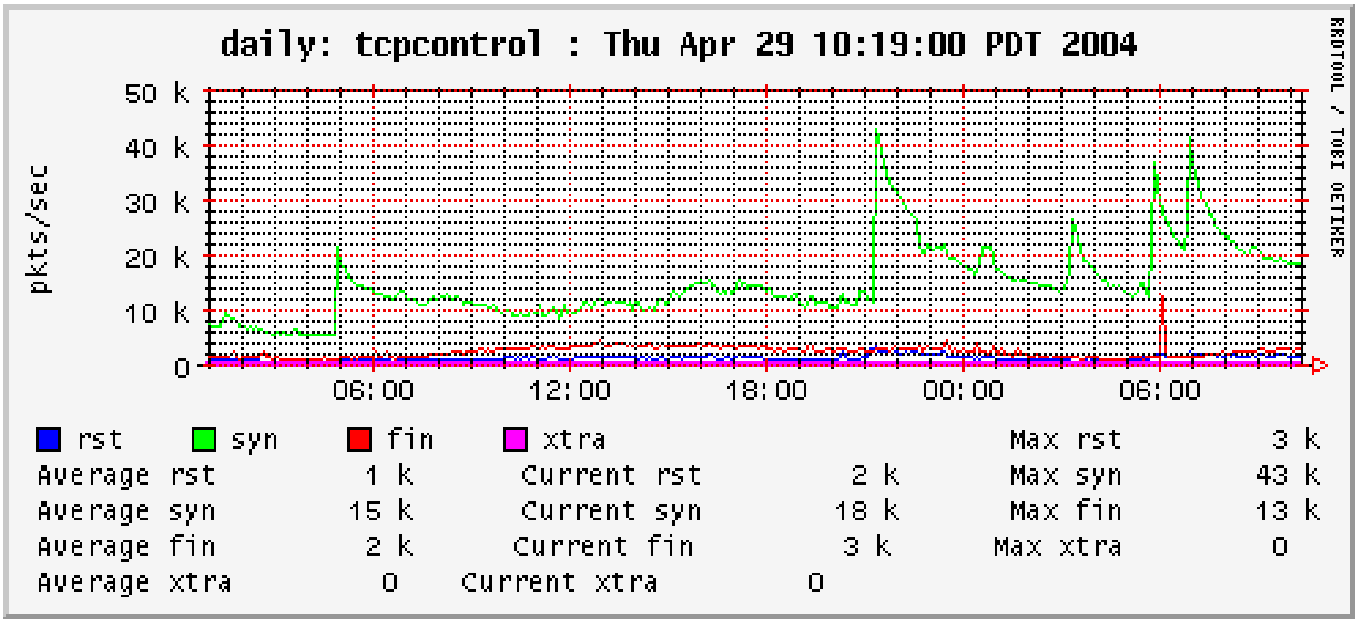

Figure 1: TCP Control Data - Large SYN Attack

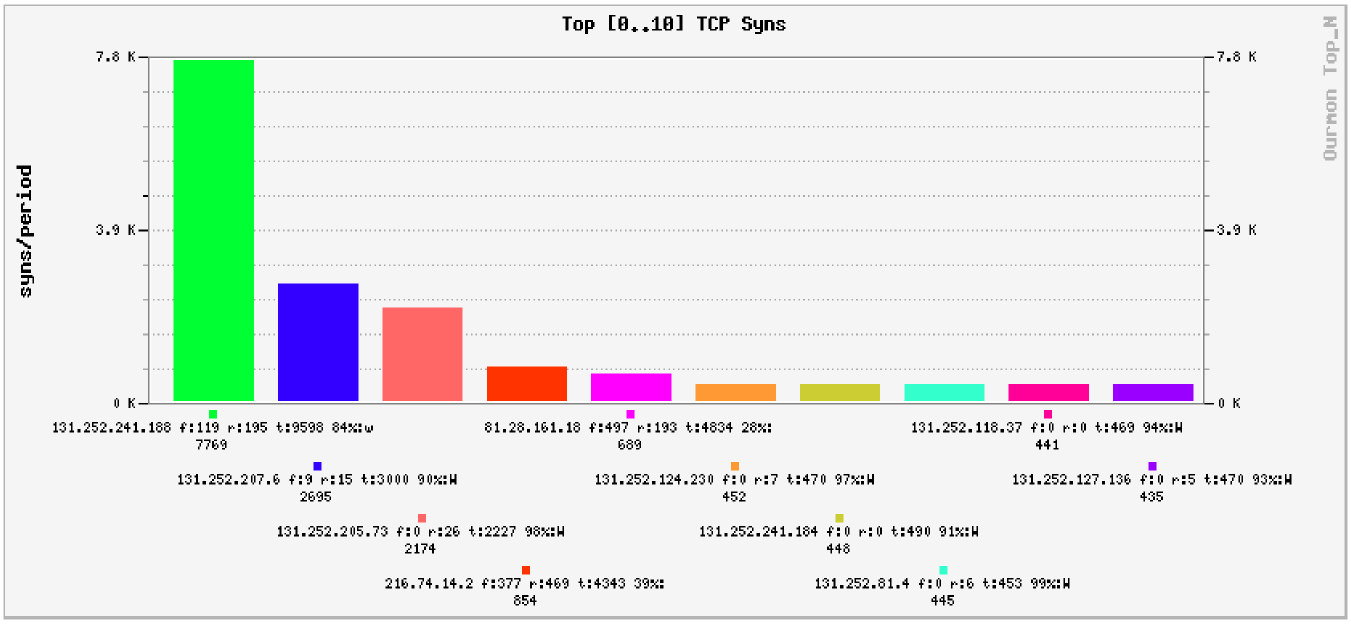

Figure 2: Top N TCP SYNS - 445 Scanner

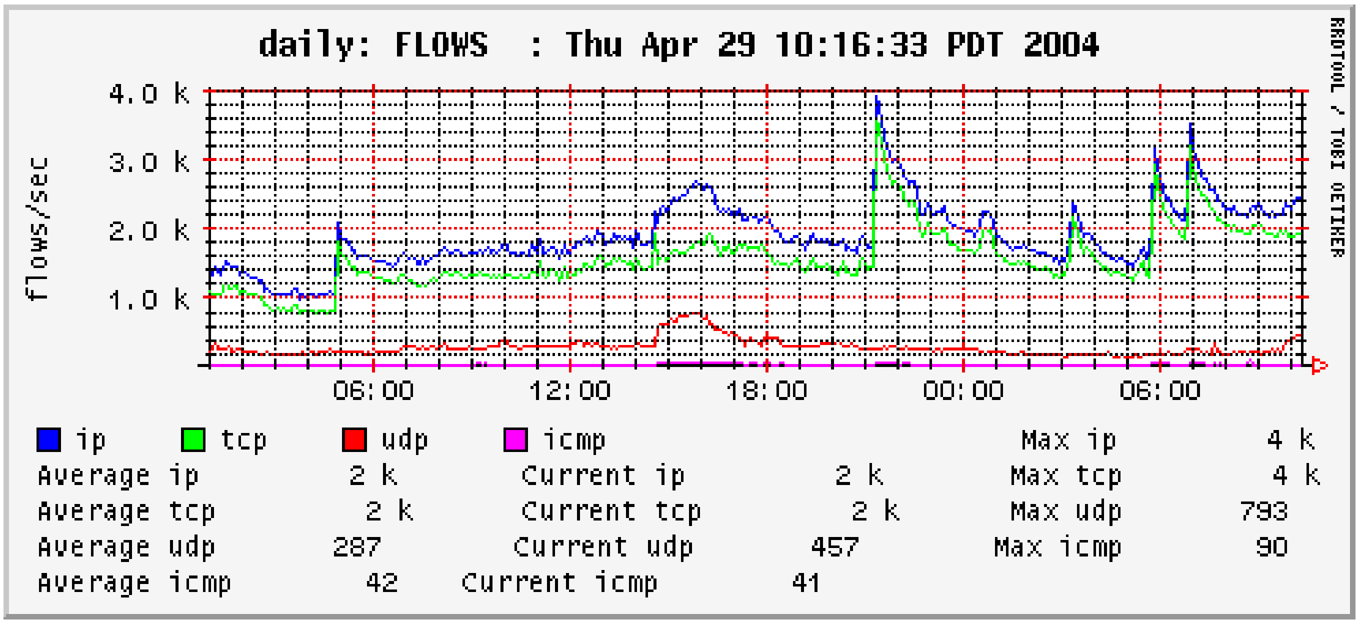

Figure 3: Top N flow count - Large SYN Attack

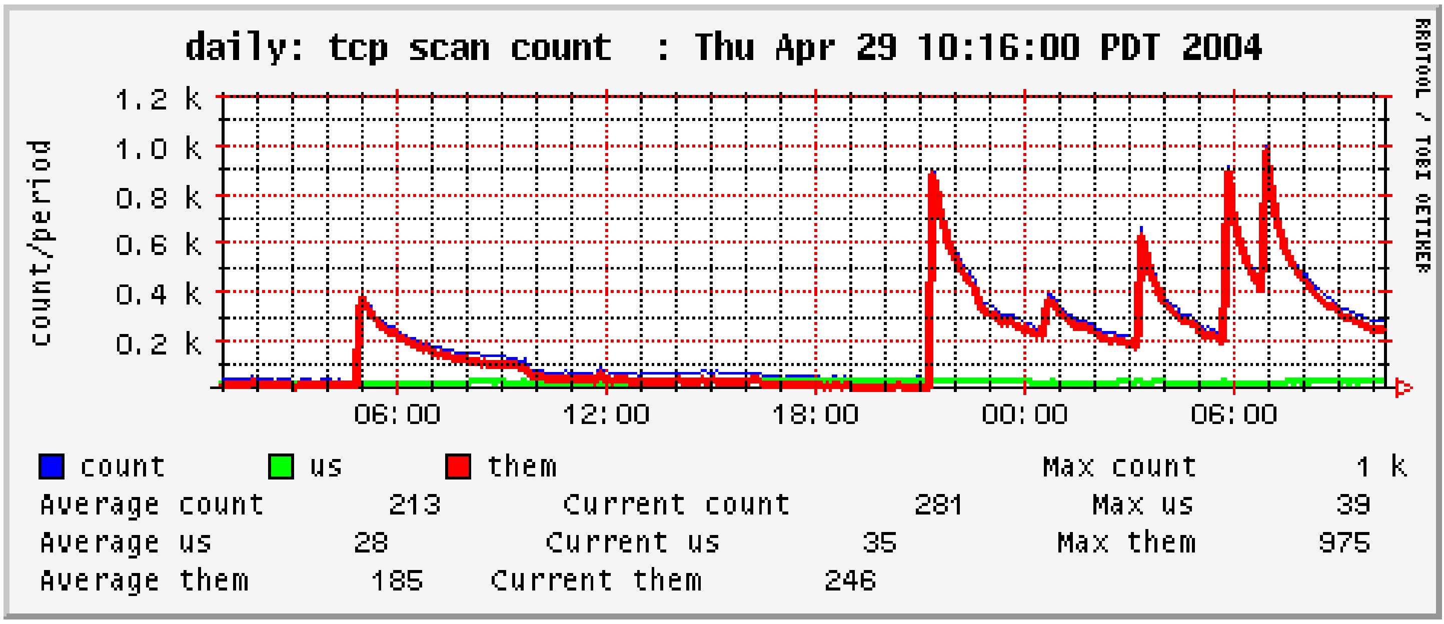

Figure 4: Worm Count - Large SYN Attack