A Framework for Real-Time Worm Attack Detection and Backbone Monitoring

Thomas Dübendorfer[*], Arno Wagner[†], Bernhard Plattner
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology, ETH-Zentrum, CH-8092 Zurich
{duebendorfer, wagner, plattner}@tik.ee.ethz.ch

Abstract

We developed an open source Internet backbone monitoring and traffic analysis framework named UPFrame. It captures UDP NetFlow packets, buffers it in shared memory and feeds it to customised plug-ins. UPFrame is highly tolerant to misbehaving plug-ins and provides a watchdog mechanism for restarting crashed plug-ins. This makes UPFrame an ideal platform for experiments. It also features a traffic shaper for smoothing incoming traffic bursts. Using this framework, we have investigated IDS-like anomaly detection possibilities for high-speed Internet backbone networks. We have implemented several plug-ins for host behaviour classification, traffic activity pattern recognition, and traffic monitoring. We successfully detected the recent Blaster, Nachi and Witty worm outbreaks in a medium-sized Swiss Internet backbone (AS559) using border router NetFlow data captured in the DDoSVax project. The framework is efficient and robust and can complement traditional intrusion detection systems.

Keywords: framework, UPFrame, plug-in, NetFlow, worm outbreak, anomaly detection, online analysis, host behaviour, Blaster, Nachi, Witty, backbone

1 Introduction

The number of security incidents each year reported by CERT/CC grew exponentially from 6 in 1988 to 137.529 in 2003 [7]. Recent massive Internet worm outbreaks such as Blaster [21], Nachi [2], Witty [24] and Sasser [22] have shown that millions of hosts [14] are patched lazily.

Monitoring traffic and detecting security problems in near real-time still seems to be only a "nice to have" (i.e. usually not implemented) capability for backbone network operators. Moreover, backbone operators currently have no monetary incentive to provide attack detection and mitigation as they get reimbursed for attack and non-attack traffic. Currently, security is mostly considered to be the responsibility of the host and access-network operators. Network-based intrusion detection systems set their focus on packet-level inspection in stub networks. These systems do not scale in high-speed networks since packet processing is extremely resource intense.

In this paper, we present our open source near real-time monitoring framework named UPFrame (pronounced "up-frame"). We explain its architecture, buffer management, plug-in support, traffic shaping algorithm, and applicability. Then we discuss several plug-ins that we developed for online monitoring of high-speed backbone traffic in order to detect worm outbreaks. We successfully used our framework and validated our plug-ins by replaying the actual Nachi and Witty worm outbreaks from our large archive of flow-level backbone traffic.

The paper is organised as follows: In Section 2, we describe flow-level backbone traffic and our DDoSVax traffic archive, which we used the real world flow-level backbone worm traffic traces from. Section 3 presents the UPFrame framework. Section 4 describes the core ideas for our detection algorithms. We demonstrate the effectiveness of our algorithms by validating the implemented plug-ins on backbone traffic from the outbreaks of several Internet worms in Section 5. The paper finishes with a discussion of the results in Section 6 and our conclusions in Section 7.

2 Flow-Level Backbone Traffic

2.1 DDoSVax traffic archive

For our observations, we used flow-level traffic data from the medium-sized Swiss Internet backbone AS559 operated by SWITCH (Swiss Academic and Research Network [1]). This backbone connects all Swiss universities and various

[*]Partially funded by the Swiss Academic Research Network SWITCH.
[†]Partially funded by the Swiss National Science Foundation under grant 200021-102026/1 and SWITCH.

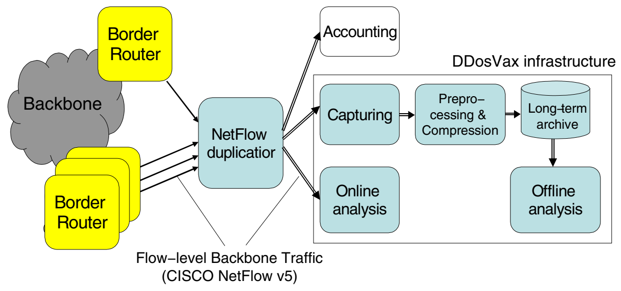

Figure 1. SWITCH network and DDoSVax infrastructure

research labs (e.g. CERN), federal technical colleges and colleges of higher education to the Internet. NetFlow data from all four AS559 border routers is captured and stored on tape for research purposes in the DDoSVax project [25]. Figure 1 shows the DDoSVax capturing infrastructure.

The SWITCH Internet Protocol (IPv4) address range contains about 2.2 million addresses. SWITCH carries around 5% of all Swiss Internet traffic [16]. On a working day, network traffic between approximately 200'000 SWITCH-internal hosts and of approximately 800'000 hosts from outside the SWITCH network can be observed. In 2004, on average 60 million NetFlow records per hour were captured, which is the full, non-sampled number of flows seen by the SWITCH border routers. The resulting data repository of roughly 6 Terabytes of bzip2 compressed NetFlow data per year, which contains the full SWITCH border flow traffic data starting early in 2003, is currently worldwide one of very few with a comparable size and level of detail.

2.2 NetFlow

In Cisco's NetFlow v5 [8] format that we use for our DDoSVax traffic archive, consecutive network packets in the same direction between the same two hosts (i.e. IPv4 addresses) using the same protocol (ICMP, UDP, TCP, others) and port numbers are reported as a single 48 bytes flow record. The number of packets, the total number of bytes in the IP layer, start and end time of the flow in milliseconds are also contained in the flow record as well as some local routing information. Our NetFlow records contain no TCP flags due to router restrictions.

3 UPFrame Framework

We were faced with the task to analyse NetFlow records exported by the SWITCH border routers in near real-time. These records arrive in bursts of UDP packets. We wanted to be able to run several algorithms in parallel on each received NetFlow packet with the option to distribute the processing load on several computers. As a result, we developed a generic application framework, named UPFrame with the core features:

Efficient capture: Receives and buffers incoming UDP packets reliably at high packet rates.

Plug-in support: Can feed the received packets to plug-ins that independently process the packets in parallel.

Traffic shaping: Buffers large amounts (megabytes) of incoming data to smoothen out data bursts. The built-in traffic shaping mechanism can control the rate of the data feed to any subscribed plug-in.

Robustness: Crashed or misbehaving plug-ins have minimal impact on overall functionality and other plug-ins. A configurable watchdog mechanism can detect and restart unresponsive or unexpectedly terminated plug-ins. It can also monitor the framework's management process.

Easy monitoring: The current operational state of the framework can be observed via a web-interface and a text-based interface suitable for automatic polling.

UPFrame was developed using C on Linux and has a size of 12'000 lines of code. It works well on Linux kernels 2.4 and 2.6 on Gentoo Linux and Debian Sarge. It has also been ported to FreeBSD 5.2.1. UPFrame is open source and was initially released in 2004. It is under the GNU GPL [4] and can be downloaded from the UPFrame web site [19].

It is noteworthy that UPFrame is not restricted to process NetFlow data packets, which it provides a parsing library for. It can process any UDP packet stream sent to a fixed port.

3.1 Architecture

Figure 2 shows a sample setup of two chained UPFrame instances. The router exports the NetFlow data as UDP packets, which arrive at the writer process of UPFrame. The writer stores these packets in shared memory segments, which are read in parallel by several plug-ins. The "UDP forward" plug-in forwards all packets, which it reads from the shared memory segments, over the network to a second instance of UPFrame on a remote computer. Likewise, a "TCP forward" plug-in sends these packets over a TCP connection to e.g. a legacy accounting system. This chaining mechanism together with the plug-in support allows a very flexible configuration. It would also be possible to send the UDP packets to several destinations (either duplicating or sampling the data) concurrently. It is possible to run the framework in multiple instances on the same machine, which can be helpful in a development setting. To enhance security, an IP source address filter can be configured that drops all UDP packets from unregistered addresses.

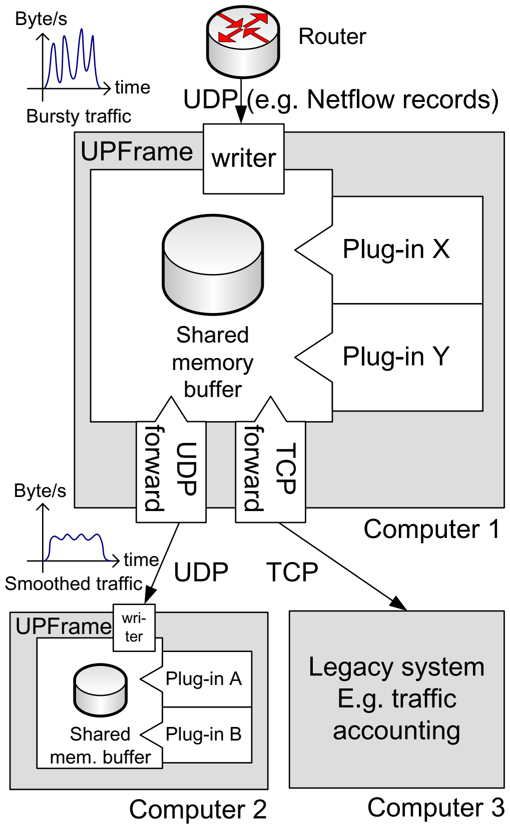

Figure 2. UPFrame architecture

3.2 Buffering

NetFlow records exported by routers typically arrive in short bursts of packets every few seconds. The burstiness is even worse if data from more than one router is captured at a single computer as the NetFlow data bursts may overlap. UPFrame prepares an internal pool of shared memory segments of a configurable maximum size. These segments are used in rather small blocks that are either in state *free*, *filled*, or *trashed* as illustrated in Figure 3. A *free* shared memory segment is waiting in a list of other free segments to be filled with data by the writer process, after which it becomes *filled*. The lists of free and filled segments managed by the memory management process are decoupled from the writer process by FIFO queues. When only few data is received by the writer and less memory is needed, the free list is reduced to a given minimum size and the superfluous segments are marked as *trashed*. After a timeout they are given back to the operating system. The plug-ins read data from a filled buffer and advance to the next newer one as soon as they are done with input processing. They can advance at their own speed (or alternatively use the traffic shaper as discussed in Section 3.3), which explains the different read positions of the plug-ins in Figure 3.

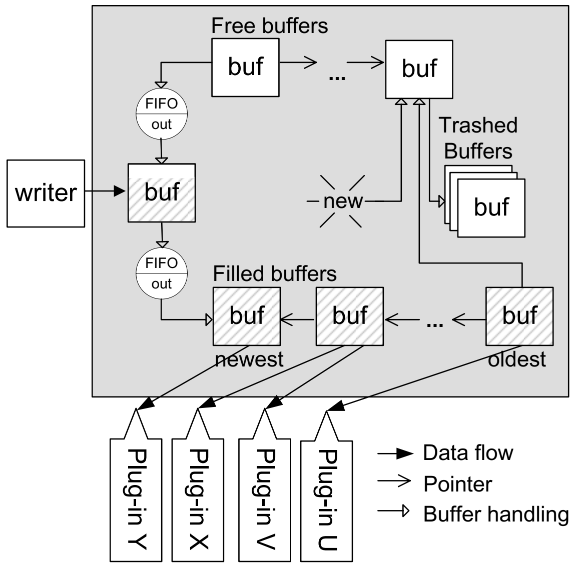

Figure 3. UPFrame buffer handling

3.3 Traffic Shaping

The traffic shaper is realized as a low pass filter on the incoming traffic rate and uses the leaky-bucket [10] principle for buffering incoming bursts. In addition, we modulate the output rate by writing out data faster if the memory buffers (the "bucket") fill up and slower if many buffers are empty. A configurable maximum output rate prevents the plug-ins from being overloaded. Mathematically, we calculate once every second

$$t_{out} = min(f_c(b) \cdot \sum_{j=1}^{n}(t_j \cdot c_j)\ ,\ t_{max})[\frac{\mu sec}{segment}]$$

with t_{out} being the current time delay after which the next filled buffer (i.e. a shared memory segment) will be fed to the plug-in (i.e. t_{out} is the inverse segment processing rate). Parameter n (e.g. n=100) is the number of past inverse input rates t_j considered. The current value of t_j is estimated once every second by averaging over the last four writes of input data to buffers (i.e. shared memory segments). This sampling helps to reduce the processing load for estimating t_j and averaging partially smoothes out input bursts. The weights c_j are used to amplify more recent behaviour and to attenuate older values. Function $f_c(b)$ returns a positive flush coefficient that exponentially increases when the current fraction b, defined as number of filled buffers not yet read by the plug-in using the traffic shaper divided by all filled buffers, raises. If b reaches 80% or more, $f_c(b)$ starts an "emergency flushing". We consider a value of 2%-5% for fraction b as optimal for normal plug-in operation based on our stress tests with real NetFlow data.

Finally, taking the minimum of t_{max} and the just calculated value limits the maximum speed for buffer-segment processing.

Each plug-in can use an individual instance of the traffic shaper by registering a call-back function for new data, which is then called according to the result of the traffic shaping algorithm. The traffic shaper and memory management were successfully validated in stress tests in a Gigabit Ethernet as documented in [18]. Figure 4 shows bursty incoming NetFlow traffic from one SWITCH router together with the target rate by the traffic shaper ("filtered") and the measured real output rate ("outgoing") of the UDP forward plug-in with activated traffic shaping. The discrepancy between the "filtered" and the "outgoing" curve is due to network socket buffering.

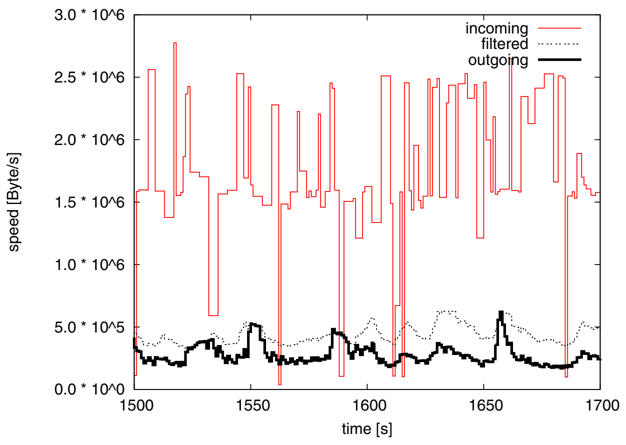

Figure 4. UPFrame traffic shaping

3.4 Plug-in Support

Each plug-in runs as a separate process. The application programming interface, realized as a library, gives access to the shared memory data buffered by UPFrame. The plug-ins either directly access the shared memory buffers through the API at their own pace or alternatively register a call-back function for UPFrame's traffic shaper mechanism. The major restrictions on the plug-ins are that they may not consume too much main memory and processing power, since they have to share these resources with the other plug-ins and the framework.

3.5 Framework Monitoring

The framework gathers statistics about warnings (e.g. when a plug-in suffers input data loss due to slow processing), buffer level, number of received and discarded packets, and others. These are accessible with our tool *stat* in human readable form that can be processed by most plot and statistics programs. There is also a watchdog, which can restart not only the plug-ins but also the framework management processes in case they crash. The performance mainly depends on the plug-ins used, the framework itself was never a bottleneck and has a very low CPU and processing overhead.

3.6 Applicability of the Framework

UPFrame was developed with the primary goal of providing a solid base for experimental and production real-time processing of backbone NetFlow data gathered in the DDoSVax project. Instead of dealing with capturing, buffer management, traffic debursting, traffic shaping, resource management, load balancing, and monitoring for failed processes, the researcher can now focus on algorithm development. Several algorithms can be run in parallel on the same input without interfering with each other. We have also developed several NetFlow tools, e.g. for replaying DDoSVax NetFlow data with the same time characteristics as during the initial capture, which allows to e.g. debug and test new algorithms in "off-line" mode.

UPFrame is an extensible light-weight framework as its name indicates and not a full-featured network monitoring system. As the framework itself does not care about the content of the captured UDP packets, it could also be used to process e.g. measurement data from temperature sensors. In Section 4, we give some sample use cases of the framework for worm outbreak detection and P2P heavy hitter identification. A test installation of the framework at SWITCH (AS559) for online network monitoring during several weeks confirmed the framework's stability. In early 2005, we monitored P2P traffic online with DDoSVax AS559 border router NetFlow data for a few months for developing and validating a heavy hitter population tracking algorithm. The validation was time critical, as we used P2P application layer polling for confirming the P2P network of a new P2P node found.

3.7 Related Work

Many NetFlow processing tools exist, commercial and non-commercial ones. Unfortunately, many of them have a very narrow focus, provide no open programming interface (especially commercial ones) or are no longer maintained. CAIDA's *cflowd* tool [6] was the first open source NetFlow capturing and processing tool released in 1998. It is no longer maintained by CAIDA. David Plonka adapted and extended *cflowd* to *Flowscan* [17], which was implemented with Perl scripts and modules and was optimised to provide near real-time traffic bandwidth usage plots with RRDTool [23] split by protocol type and port. No development seems to have happened after 2003. Mark Fullmer's *flow-tools* [11], last updated in March 2003, are a collection of NetFlow tools for capturing, storing, filtering and reporting. Some packet based network monitoring systems such as ntop [5] also provide NetFlow support. The nProbe extension of ntop can act as NetFlow aggregator that emits Cisco-like NetFlow records from packet captures or calculate some basic statistics on the received flows. Those packet-based tools were developed and optimised for monitoring local area networks and not backbones.

4 Worm Detection Plug-ins

Even though high-speed Internet links have become a commodity, only little is known about the actual host behaviour in large networks. Network operators often merely count the total traffic that they transport for their customers as they need it for accounting reasons, possibly split by the most important protocols (TCP, UDP, ICMP, other) as well as some well-known services (e.g. HTTP, SMTP). When it comes to security incidents, some operators of larger networks do forensic analyses on captured flow-level data. However, they need to know exactly what to look for.

We developed several algorithms for host behaviour, network activity and traffic characterisation and implemented them as plug-ins for UPFrame. These plug-ins are able to process incoming NetFlow data from the SWITCH border routers in near real-time and store a log of the calculated traffic statistics. A web server (not part of UPFrame) with interactive scripts provide a graphical user interface for the network operator to monitor traffic by using the plug-ins' statistics and visualised output data of a given point in time or a time range.

In the following, we describe the core ideas behind our plug-in algorithms and why we think that they detect interesting anomalies in the backbone traffic. Later in this paper, we will apply them to monitor the outbreak of large-scale Internet worms.

4.1 Host Behaviour Based Detection

Our host behaviour based analysis is described in full detail in [9]. It assigns to each host a set of behaviour classes within each one minute time interval. These classes are defined such that a host becomes a member only if it behaves unusual. A rapid change in the number of hosts in any class indicates a significant change in the network behaviour of the observed hosts. Our assumption is that if hosts try to infect others during a worm outbreak, the behaviour of many hosts will change in a similar way.

We define three behaviour classes by threshold conditions that the traffic of an observed host must satisfy in order to be member of that class. The values given in brackets were used for the host behaviour plots in this paper. They were optimised for making our behaviour based detection sensitive to various major worm outbreak events (see also [9]) by replaying DDoSVax NetFlow data to the plug-in with different parameter settings and comparing the plug-in outputs for most significant changes.

- **traffic class**: $\frac{\text{Bytes sent}}{\text{Bytes received}}$ (>3)
- **connector class**: # outgoing connections (>10)
- **responder class**: # bidirectional connections (>1)

We define a bidirectional connection as a pair of flows in opposite direction between a pair of hosts, where the start times of the flows fall withing an interval of less than 50 ms. To accommodate large volumes of NetFlow data and to assure real-time operation, a filter stage was prepended to the plug-in, that filters the flows by protocol type (TCP, UDP, ICMP single or combined).

The algorithm keeps two one minute time interval buckets to sort the incoming flows. For each interval, a hash table stores the the hosts seen together with the parameters amount of traffic sent and received, and the number of outgoing connections. Bidirectional connections are handled with nested hash tables to achieve fast lookups and to minimise memory requirements. The source host hash table stores the source IP address for each observed host. A lookup of such a host returns a hash table with all flows originating at this host that were seen so far in the current interval. An efficient lookup by a hash key of destination IP address, source and destination port in this hash table is enough to match a new flow with an existing one in the opposite direction.

An upper memory limit for this plug-in can be set. If the plug-in needs more memory than the limit, it discards new flows for the current interval and sets an appropriate error code for the interval in its output data. When flows for the next time interval arrive, the algorithm resumes its operation. This ensures that the plug-in can deal with a large-scale attack that it was not designed for. The plug-in can also deal with missing data (e.g. due to an interruption of incoming NetFlow records) and will automatically start a new time interval if it suddenly receives NetFlow records that carry timestamps outside the currently observed time intervals.

4.2 Activity Based Detection

The activity plug-in tracks the network activity of all monitored Internet hosts. The activity of Internet hosts could be visualised by plotting each of the 2^{32} possible IPv4 host addresses in an image of 4'294'967'296 pixels. Each pixel representing a single host is coloured depending on the traffic amount a host has sent or received in a time interval. As a square image of 65'536 pixels on each side is too large in most settings, we let the user restrict the range of the IP addresses displayed in the image and we add the possibility to group neighbouring IP addresses (as "virtual subnet" with /X bit subnet prefix) together and show their total traffic with a single pixel. This virtual subnetting allows to interactively zoom into subnetworks of interest with a resolution of up to one pixel per host through an interactive image map in the web interface of the plug-in.

The activity maps of Internet hosts in Figure 6 (TCP traffic) and Figure 8 (incoming ICMP traffic) are organised in stripes of neighbouring IP addresses as shown in Figure 5. If hosts with neighbouring addresses are very active, they can be seen as a line of lighter pixels. Black pixels indicate no traffic at all. The number of bytes for each flow is accounted once to the sending host as outbound and once to the receiving host as inbound.

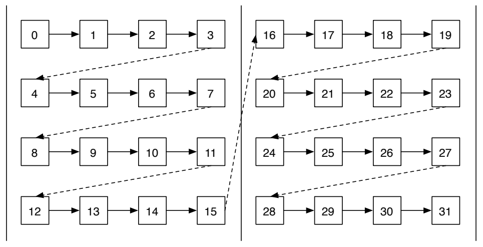

Figure 5. The IP activity plot shows stripes of neighbouring IP addresses

Regarding memory consumption, a naïve approach would use an array to hold the traffic amount for each of the 2^{32} possible IPv4 addresses in two 4 byte integers, yielding a memory footprint of $2^{32} * (4 + 4) = 32$ GB. As this would be problematic in main memory, we use a hash table and store only the traffic of observed hosts with activity in the currently processed intervals.

4.3 Further Plug-ins and Analyses

We also implemented algorithms for filtering and counting the number of flows for a given TCP or UDP source or destination port, and for counting the number of unique IP addresses using a service specified by ports and protocol as shown in Figure 7. They are well suited to track down worm activity as soon as the traffic characteristics of a new worm are known. Due to the huge range of possible filter criteria such port usage filters are less suited for the detection of unknown worms. We developed another generic approach for worm outbreak detection based on compressibility of certain network traffic parameters. In Section 5.2.2, we illustrate this method on the Witty worm outbreak. Experimental plug-ins for tracking P2P node activity for the most common P2P networks, as well as for the analysis of IRC traffic between possible malicious IRC bots were also developed. The large range of possible analyses shows that the framework is a versatile platform for exploring new algorithms on real traffic data. We use UPFrame mainly for research with real-time and replayed NetFlow data. Since begin of 2005, network services at ETH Zurich use UPFrame 24/7 with a plug-in for monitoring P2P heavy hitters [15].

5 Analysis of Real Worm Traffic

5.1 Nachi Worm

The Nachi worm [2], also known as Welchia worm [3], was an attempt to use a worm against a worm infection. It was first observed on August 12th, 2003, around 6:00 UTC. Nachi exploited the same vulnerability as the original Blaster worm, namely a DCOM RPC vulnerability on port 135/TCP of hosts running Windows XP. In addition, Nachi also exploited a vulnerability in WebDAV on port 80/TCP found in Microsoft IIS 5.0. The second exploit is believed to mainly impact hosts running Windows 2000. Unlike Blaster, the Nachi worm uses an ICMP echo request (a.k.a. "ping") to determine whether a specific IP address is in use. This caused massive ICMP activity. We observed that during the Nachi outbreak as much as 6% of all packets and as much as 1% of the total traffic volume in the SWITCH network were caused by Nachi ICMP messages. Nachi also seemed to have been fairly unsuccessful in stopping Blaster and instead added to the overall damage.

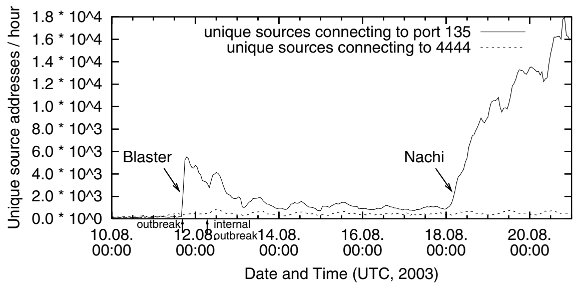

Figure 7. Blaster and Nachi worm outbreaks

5.1.1 Activity Based Analysis

Amazingly, the number of unique source IP addresses, from which traffic to port 135/TCP originates per hour was more than three times higher while Nachi was active compared to during the Blaster outbreak as can be seen in Figure 7. The sudden and intensive use of the formerly rarely used RPC service can be clearly noted as a sign of anomalous host behaviour.

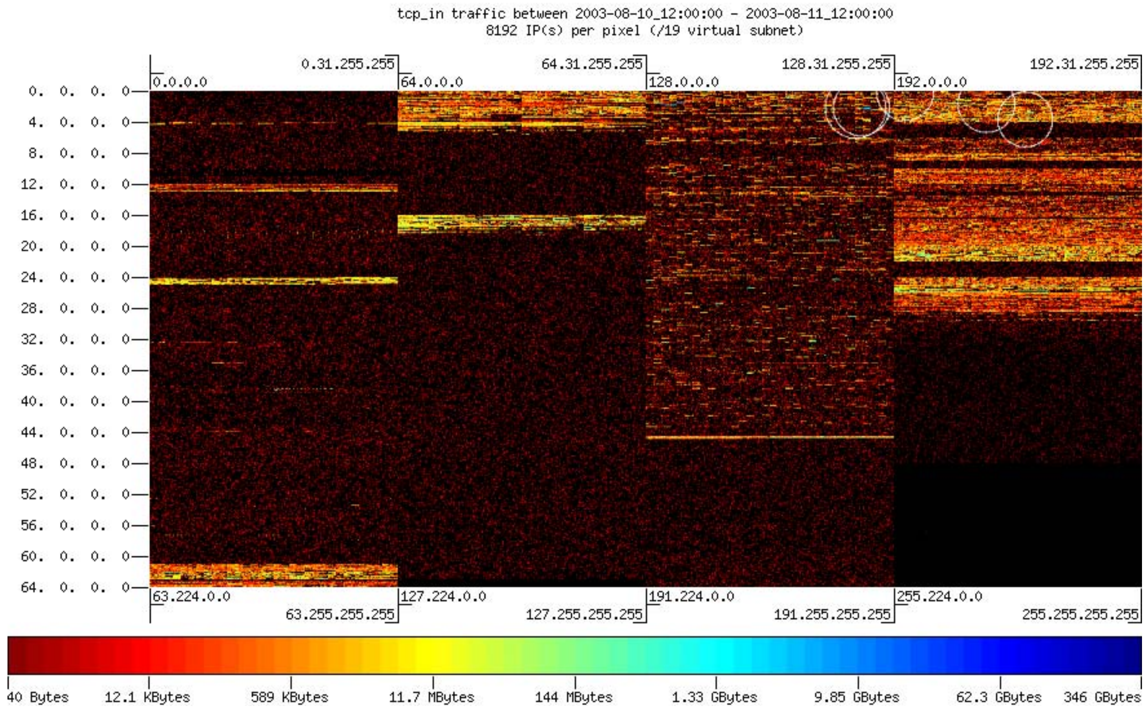

Figure 6. Total TCP inbound traffic for one day as seen in the SWITCH network. The 5 most active hosts are highlighted with a white circle.

The very characteristic ICMP activity pattern of Nachi can be observed in Figure 8. It is striking that IP addresses containing the byte 0xC5 (=197 decimal) are not scanned, which results in black lines in our ICMP activity plot (see white arrows in figure). This can be easily explained according to an article in the Virus Bulletin [12] by the fact that for both of Nachi's exploits, RPC DCOM and WebDAV, it needs to patch shell code containing overlong paths with the bytes of the IP address XOR-ed with 0x99. As 0xC5 XOR 0x99 = 0x5C and character 0x5C would be interpreted as a backslash "\" character, the worm needs to avoid using it when patching the transmitted shell code. Such irregular scanning activity is particularly useful to nail down the cause of such ICMP scanning traffic. Such scanning patterns involving many target networks can almost exclusively be observed within a larger network.

5.2 Witty Worm

The Witty worm [20, 24] was first observed in the wild on Saturday, March 20th, 2004, at approximately 4:45 UTC. It exploits a bug in several products by Internet Security Systems ISS [13]. The Witty exploit is UDP based. It differs from most other exploits insofar as it uses a randomised target port and the fixed source (!) port 4000/UDP. The vulnerable host population for Witty was around 12'000 hosts. Witty was the first fast worm that demonstrated that even such a small vulnerable population can be infected in a matter of hours. Witty carries a destructive payload that causes random data loss on the hard disk.

5.2.1 Host Behaviour Based Analysis

The Witty hosts, which involved many new hosts opening connections with the fixed source port 4000/UDP shows up clearly in the connector class plot of Figure 9. Around noon, it seems that for almost 3 hours a filter was installed, which was deactivated later.

The comparison of the connector class cardinality of the "Witty day" with the regular "Witty-free" previous Saturday shows the worm outbreak even clearer.

5.2.2 Compressibility Analysis

The Witty worm can also be reliably detected by tracking the compressibility ratios of certain network traffic parameters like IP source address, IP destination address, source port, or destination port. Our compressibility detection method is described in more detail in [26]. We store a sequence of the same packet header parameter (e.g. IP address) over a 5 minute interval in an array. Then we compress this array using the lzo algorithm and find the ratio of

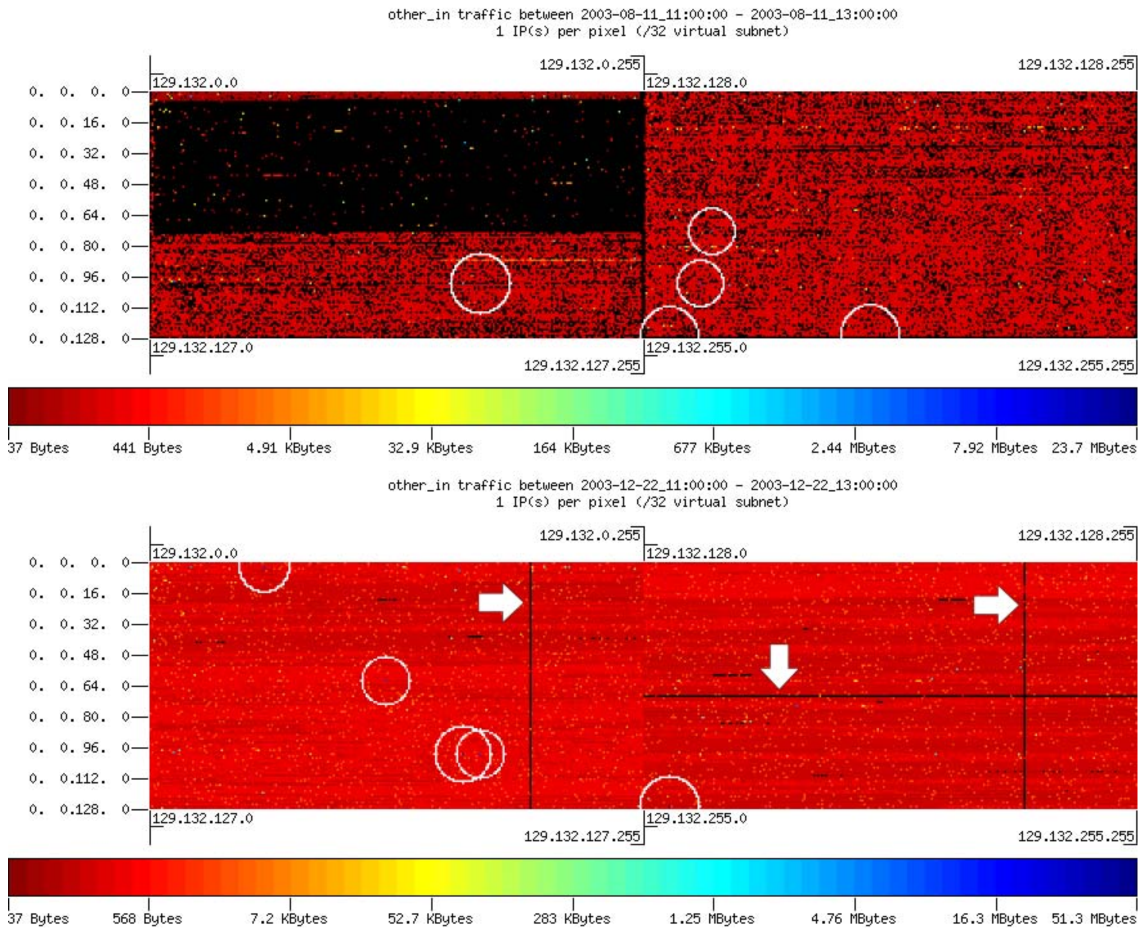

Figure 8. Normal vs. Nachi ICMP incoming network activity

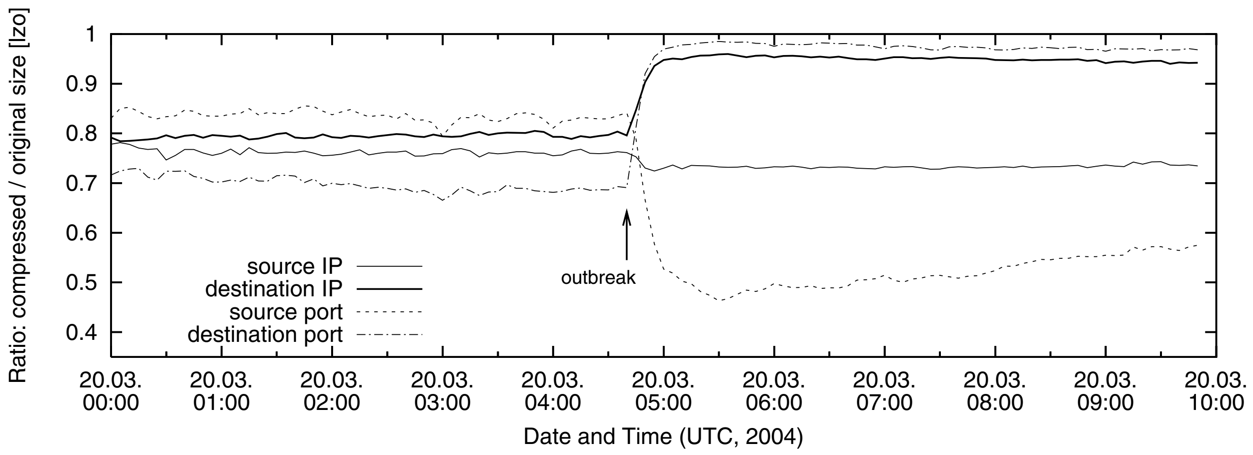

Figure 10. The Witty outbreak clearly changes the compressibility-ratio graphs

compressed vs. original data size. This ratio is then plotted over time. In Figure 10, one can clearly see that after the outbreak of Witty, the source ports become significantly better compressible. This is caused by the fact that many Witty UDP packets carrying the same source port 4000 can be observed in the network. At the same instant, the des-

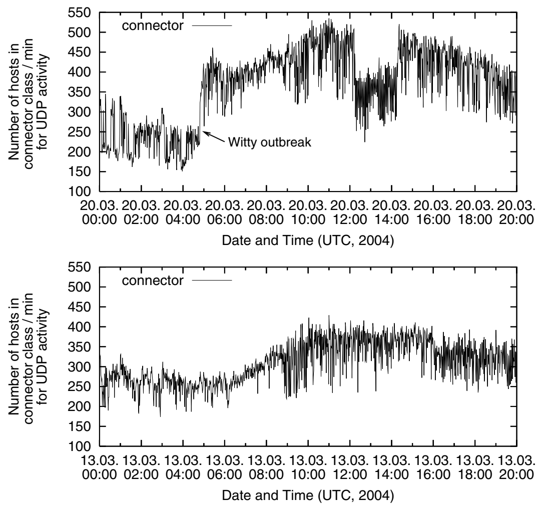

Figure 9. The Witty outbreak significantly raises the number of connecting hosts compared to the previous Saturday.

tination ports and IP addresses become less compressible. This is due to randomly chosen destination ports and IP addresses in the Witty packets by infected hosts.

6 Discussion

Classical IDS systems cannot be used for monitoring high-speed networks, since they lack the needed performance and are not well suited for new threats such as fast worms. In addition they often require packet payload information, which is in most cases not available in backbone networks and other fast networks.

We have shown in this paper that meaningful monitoring of high-speed networks in near real-time is possible with relatively low effort and based only on flow-level data. In contrast to packet payloads, aggregated header information in the form of flow records such as Cisco's NetFlow is usually available, since it is used for accounting and general network load monitoring. While monitoring as we described it in this paper may today be viewed as "nice to have" by many network operators, it seems reasonably to expect it to grow into a necessary element for successful backbone operations in the future.

While this paper mainly focuses on attack patterns generated by outbreaks of fast worms, we believe that also other types of events can be observed with comparable effort on flow-level.

7 Conclusions

Our open source framework UPFrame [19] and its network traffic monitoring plug-ins can help network service providers to better monitor and react to anomalous network activity in fast backbones with large traffic volumes. Having near real-time network backbone traffic analyses ready improves security as it cuts down reaction time in case of massive network attacks. Mere forensic analyses are no longer sufficient for securing the Internet. We would like to enable network operators to better know and understand the current traffic in their network and to give researchers a tool for exploring new worm and attack detection algorithms.

Our multi-paradigm approach to analyse traffic with a multitude of efficient algorithms concurrently has proven successful in catching anomalous traffic behaviour at the outbreak of and during massive network attacks. The impact and the clear visibility of real Internet worms like Blaster, Nachi and Witty in high-speed backbone network traffic were illustrated in various plots.

We want to emphasise that UPFrame can be used not only for security related work, but also to gain better insights into general network usage patterns. Future work will be directed in further elaborating on anomaly detection plug-ins and to broaden the scope of algorithms tested on real attacks using the ETH DDoSVax NetFlow archive of SWITCH border router traffic.

8 Acknowledgements

We want to thank our students Caspar Schlegel (UPFrame), Andrea Weisskopf (Activity plug-in), and Bernhard Tellenbach (Host behaviour plug-in) for their contributions to the DDoSVax project. We also want to thank Simon Leinen and Peter Haag at SWITCH for the possibility to install UPFrame in their network for stress tests.

References

[1] The Swiss Education & Research Network. `http://www.switch.ch/`.
[2] McAfee: W32/Nachi.worm. `http://vil.nai.com/vil/content/v_100559.htm`, August 2003.
[3] W32.Welchia.Worm. `http://securityresponse.symantec.com/avcenter/venc/data/w32.welchia.worm.html`, August 2003.
[4] GNU GPL. `http://www.gnu.org/licenses/gpl.html`, Jan. 2005.
[5] ntop - a network traffic probe. `http://www.ntop.org/`, June 2005.
[6] CAIDA. cflowd tools. `http://www.caida.org/tools/measurement/cflowd/`, 1998.
[7] CERT/CC. Statistics 1988-2003. `http://www.cert.org/stats/cert_stats.html`.
[8] Cisco. White Paper: NetFlow Services and Applications. `http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm`, 2002.
[9] T. Dübendorfer and B. Plattner. Host Behaviour Based Early Detection of Worm Outbreaks in Internet Backbones. In *Proceedings of 14th IEEE WET ICE / STCA security workshop*. IEEE, June 2005.

[10] P. Ferguson and G. Huston. Quality of Service: Delivering QoS on the Internet and in Corporate Networks. `http://en.wikipedia.org/wiki/Leaky_bucket`, 1998.

[11] M. Fullmer. flow-tools. `http://www.splintered.net/sw/flow-tools/`, Mar. 2003.

[12] H. Gabor Szappanos VirusBuster. Virus Bulletin: Virus information and overview - W32/Welchia. `http://www.virusbtn.com/resources/viruses/welchia.xml`, Apr. 2004.

[13] ISS. Internet Security Systems. `http://www.iss.net/`, 2004.

[14] R. Lemos. MSBlast epidemic far larger than believed. `http://news.com.com/MSBlast+epidemic+far+larger+than+believed/2100-7349_3-5184439.html`, 2004.

[15] Lukas Haemmerle. Master's Thesis: P2P Filesharing Population Tracking. `ftp://www.tik.ee.ethz.ch/pub/students/2004-So/MA-2004-04.pdf`, 2005.

[16] O. Müller, D. Graf, A. Oppermann, and H. Weibel. Swiss Internet Analysis. `http://www.swiss-internet-analysis.org/`, 2004.

[17] D. Plonka. Flowscan. `http://www.caida.org/tools/utilities/flowscan/`, Oct. 2003.

[18] C. Schlegel. Diploma thesis: Realtime UDP NetFlow Processing Framework. `ftp://www.tik.ee.ethz.ch/pub/students/2003-So/DA-2003-26.pdf`, September 2003.

[19] C. Schlegel and T. Dübendorfer. UPFrame - A generic open source UDP processing framework. `http://www.tik.ee.ethz.ch/~ddosvax/upframe/`, January 2005.

[20] C. Shannon and D. Moore. CAIDA: The Spread of the Witty Worm. `http://www.caida.org/analysis/security/witty/`, March 2004.

[21] Symantec Security Response. W32.Blaster.Worm. `http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html`, 2003.

[22] Symantec Security Response. W32.Sasser.Worm. `http://securityresponse.symantec.com/avcenter/venc/data/w32.sasser.worm.html`, April 2004.

[23] Tobi Oetiker. RRDTool - A round robin database tool for logging and graphing time-series data. `http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/`.

[24] US-CERT. Vulnerability Note: Witty (VU#947254). `http://www.kb.cert.org/vuls/id/947254`, 2004.

[25] A. Wagner, T. Dübendorfer, and B. Plattner. The DDoSVax project at ETH Zürich. `http://www.tik.ee.ethz.ch/~ddosvax/`, 2005.

[26] A. Wagner and B. Plattner. Entropy Based Worm and Anomaly Detection in Fast IP Networks. In *Proceedings of 14th IEEE WET ICE / STCA security workshop*. IEEE, June 2005.