Doctoral Thesis

Impact analysis, early detection and mitigation of large-scale internet attacks

Author(s): Dübendorfer, Thomas

Publication Date: 2005

Permanent Link: https://doi.org/10.3929/ethz-a-005113607

Rights / License: In Copyright - Non-Commercial Use Permitted
Impact Analysis, Early Detection and Mitigation of Large-Scale Internet Attacks

A dissertation submitted to the SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZÜRICH for the degree of Doctor of Sciences

presented by THOMAS P. DÜBENDORFER Dipl. Informatik-Ing. ETH born May 1st, 1975 citizen of Bassersdorf (ZH), Switzerland

accepted on the recommendation of Professor Dr. Bernhard Plattner, examiner Dr. Tom Longstaff, co-examiner Prof. Dr.-Ing. habil. Wolfgang Kröger, co-examiner

2005
Abstract

Internet security development lags far behind the unprecedented rapid growth of the Internet, which is currently approaching one billion users worldwide. Network operators and companies that rely on the Internet for their daily business are challenged continuously by numerous newly discovered vulnerabilities of their Internet hosts, servers, routers and services. As more and more cyber criminals join the Internet, intensity and frequency of large-scale Internet attacks grow at a discomforting pace.

Currently, economic damage of Internet attacks and spam is mostly ignored, only little is publicly known about the real impact of recent network security incidents, and no comprehensive and efficient early attack detection and mitigation mechanisms exist. This PhD thesis tackles these core problems by contributing an economic damage model for Distributed Denial of Service (DDoS) attacks, in-depth analyses of the past Blaster Internet worm, a software framework for online algorithm research, a method for online host behaviour based worm outbreak detection using flow-level backbone traffic and a fundamental concept for building a safe distributed traffic control system that can be used to mitigate large-scale Internet attacks.

We developed an economic damage model, which is the first public comprehensive model that transparently estimates financial damage caused by DDoS attacks suffered by Internet-dependent organisations and entire nations. With our model we estimated that if the whole of Switzerland is affected by a massive DDoS attack that results in an Internet blackout lasting one week, the economic damage to the Swiss economy with an annual GDP of CHF 482 billion sums up to CHF $6 billion, i.e. $1.2\%$ of GDP. Interviewed companies were currently found to be only vaguely
aware of the possibly huge impact of losing Internet connectivity for longer periods. With our model we hope to raise the awareness of many companies and policy makers of economies about their strong and still growing Internet dependence. We also present a method for estimating economic damage caused by spam and virus infected e-mails.

In order to investigate real Internet attack traffic, we cooperated in the DDoSVax project at ETH Zürich with the medium-sized Swiss backbone provider SWITCH (AS559). This gave us access to a comprehensive flow-level (NetFlow) traffic archive, which contains all major worm outbreaks and attacks that took place since spring 2003. Using this wealth of information, we performed in-depth forensic analyses of the network worm Blaster.A and the e-mail worm Sobig.F. We found that while exploit code transfer of the multi stage worm Blaster.A was quite frequent, actual successful infections and worm code transfers over the backbone border routers were surprisingly few. For Sobig.F we revealed among other things the timing characteristics of the immense flood of infected e-mails it created and that its greedy spreading behaviour caused many packet retransmissions.

Insights gained from these analyses lead to the design of generic near real-time online algorithms that use flow-level backbone traffic as input. We developed a new classification of hosts regarding their Internet usage behaviour based on three attack sensitive traffic features. By applying this classification to hosts before and during worm outbreaks, we showed that by tracking the number of members in these classes and class combinations not only network worms like Witty and Blaster but also e-mail worms like Sobig.F and MyDoom.A can be detected early. Using it for an early warning system can help network operators to counteract attacks earlier and more effectively. Our software framework named “UPFrame”, which we used to test and run our online algorithm plug-ins, was released as open source software.

For comprehensive and more effective cyber attack countermeasures, adding more flexibility to the Internet core plays a key role. Ideas like active or adaptive networking were around for many years. However, the hesitation was strong by network operators to incorporate any such technologies as data privacy issues and the danger of losing control over the network were not adequately addressed. We proposed a fundamental concept of “traffic ownership”, which has the potential to diminish if not eliminate such concern. Finally, we designed a novel Internet traffic
control system that incorporates our fundamental concept and allows a safe delegation of specific traffic control features from network operators to network users. Our system enables many previously unthinkable new network-integrated services. In medium term, we expect a shift of the responsibility for enforcing Internet security from network users to network operators. The possibility for new security services integrated into the Internet core will foster this trend. Assured by our research results, we think that incorporating more security within the Internet core is feasible, effective and reasonable.
Zusammenfassung

Die Sicherheit im Internet hat sich wesentlich langsamer entwickelt als das unvorhergesehene gewaltige Wachstum des Internets, das in Kürze weltweit eine Milliarde Benutzer erreichen wird. Netzwerkbetreiber und Firmen, welche für ihr Tagesgeschäft auf das Internet angewiesen sind, werden herausgefordert durch die fortlaufende Entdeckung zahlreicher neuer Schwachstellen bei Computern, Servern, Routern und Diensten im Internet. Da immer mehr Kriminelle im Internet auftauchen, steigen die Intensität und die Häufigkeit grosser Attacken mit einer beängstigenden Geschwindigkeit.


Wir haben ein wirtschaftliches Schadensmodell entwickelt, welches das erste solche umfassende und öffentliche Modell ist, das transparent den finanziellen Schaden von DDoS Attacken berechnet, der von Internetabhängigen Firmen oder ganzen Nationen erlitten wird. Eine Schätzung mit unserem Modell ergab, dass im Falle einer massiven DDoS Attacke


Contents

1 Introduction ................................. 1
   1.1 Impact Analysis ........................... 3
   1.2 Early Detection ............................ 3
       1.2.1 Introduction to Attack Detection ...... 4
       1.2.2 The DDoSVax Project ................. 4
       1.2.3 Flow-Level Worm Traffic Analysis .... 4
       1.2.4 A Framework for Real-Time Worm Attack Detection and Backbone Monitoring ... 5
       1.2.5 Host Behaviour Based Early Detection of Worm Outbreaks in Internet Backbones ... 6
   1.3 Mitigation ................................ 7
   1.4 Research Contributions .................... 8

II Impact Analysis ............................... 11

2 Economic Damage Model ....................... 13
   2.1 Introduction ............................... 13
       2.1.1 Outline .............................. 14
       2.1.2 Related Work .......................... 14
       2.1.3 Approach ............................. 15
       2.1.4 Internet System Model ................. 16
       2.1.5 Threat Model and User Impact .......... 17
       2.1.6 Limitations of Our Damage Model ...... 17
II Early Detection 37

3 Attack Detection Approaches 39

3.1 Benefits of Early Detection 39
3.2 Data Sources for Detection 40
3.3 Detection Methods 41
3.3.1 Signature Matching 41
3.3.2 Traffic Anomaly Analysis 42
3.3.3 Black Hole or Darknet Monitoring 43
3.4 Outline 44
4 The DDoSVax Project 45
  4.1 DDoSVax NetFlow Archive 46
  4.2 SWITCH Internet Backbone 46
  4.3 DDoSVax Infrastructure 49
    4.3.1 Cluster Scylla 50
  4.4 NetFlow 52

5 Blaster and Sobig Analyses 55
  5.1 Introduction 55
    5.1.1 Backbone Measurement Data 56
    5.1.2 Related Work 56
  5.2 Network Worm Blaster 59
    5.2.1 Outbreak 59
    5.2.2 Worm Variants 59
    5.2.3 Blaster’s Infection Steps 60
    5.2.4 Identification of Blaster Infections by Flows 62
    5.2.5 Blaster Outbreak Traffic Analysis 64
    5.2.6 Worm Code of Multi Stage Worms:
        Low Frequency vs. High Threat Potential 71
    5.2.7 Coarse Grained Analysis 72
  5.3 E-Mail Worm Sobig.F 79
    5.3.1 Outbreak of Sobig.F 79
    5.3.2 Identification of Sobig.F E-Mails 80
  5.4 Conclusions and Outlook 83

6 UPFrame 87
  6.1 Introduction 87
  6.2 UPFrame Framework 88
    6.2.1 Architecture 89
    6.2.2 Buffering 90
    6.2.3 Traffic Shaping 92
    6.2.4 Plug-in Support 93
    6.2.5 Framework Monitoring 94
    6.2.6 Applicability of the Framework 94
    6.2.7 Related Work 95
6.3 Worm Detection Plug-ins ........................................ 96
   6.3.1 Activity Based Detection .............................. 96
   6.3.2 Port Usage Statistics ................................ 98
   6.3.3 Further Plug-ins and Analyses ....................... 98
6.4 Analysis of Real Worm Traffic .............................. 99
   6.4.1 Nachi Worm ........................................ 99
   6.4.2 Witty Worm ......................................... 102
6.5 Discussion ............................................. 103
6.6 Conclusions and Outlook .................................. 104

7 Behavioural Worm Detection ................................. 105
   7.1 Introduction .......................................... 105
   7.2 Related Work ......................................... 106
   7.3 Host Behaviour Based Worm Detection .................... 107
     7.3.1 Method ........................................ 107
     7.3.2 Implementation ................................ 108
   7.4 Validation on E-Mail Worms .............................. 110
     7.4.1 Sobig.F ......................................... 110
     7.4.2 MyDoom.A ...................................... 114
   7.5 Validation on Network Worms ............................. 118
     7.5.1 Witty Worm ...................................... 118
     7.5.2 Blaster Worm .................................... 118
   7.6 Conclusions and Outlook ................................ 120

8 Early Detection Conclusions .................................. 121

III Mitigation .................................................. 123

9 Traffic Control Service ....................................... 125
   9.1 Introduction .......................................... 125
   9.2 Attack Scenarios ...................................... 127
     9.2.1 Distributed Denial of Service Attacks ........... 127
     9.2.2 DDoS Reflector Attacks .......................... 128
   9.3 Analysis of Mitigation Mechanisms ...................... 129
## Contents

9.3.1 Reactive Mitigation Strategies .... 130  
9.3.2 Proactive Mitigation Strategies .... 131  
9.3.3 Discussion of Mitigation Effectiveness .... 132  
9.4 Distributed Traffic Control by IP Address Owners .... 133  
9.4.1 Network Traffic Control Service .... 133  
9.4.2 Traffic Processing Device Functionality .... 134  
9.4.3 Attack Prevention and Defence .... 135  
9.4.4 Emerging Applications .... 136  
9.4.5 Security Considerations .... 137  
9.4.6 Incentives for Deployment .... 138  
9.5 Infrastructure .... 139  
9.5.1 Network Model .... 139  
9.5.2 Traffic Control Unit Architecture .... 142  
9.5.3 Traffic Control Unit Placement .... 143  
9.5.4 Traffic Control Unit Filter Efficiency .... 146  
9.5.5 Scalability Issues .... 148  
9.6 Use Case .... 150  
9.6.1 DDoS Mitigation Service .... 150  
9.6.2 Service Deployment .... 152  
9.7 Conclusions and Future Work .... 155  

10 Conclusions and Outlook .... 157  
10.1 Conclusions .... 157  
10.2 Outlook .... 161  

Acknowledgements .... 163  
Curriculum Vitae .... 165  
A NetFlow v5 Format .... 167  
Bibliography .... 170
## List of Figures

2.1 Internet system model ........................................ 16
2.2 Loss of a BSP, an ISP and a web service provider ........ 24
2.3 Loss of a corporate customer .............................. 25
2.4 Loss of an insurance company .............................. 27
2.5 Loss of a telco .................................................. 28

4.1 The SWITCH academic and research backbone (AS559) . 48
4.2 DDoSVax infrastructure ........................................ 49
4.3 Front and back view of cluster Scylla ...................... 51

5.1 Blaster's infection steps ....................................... 60
5.2 Blaster outbreak: Number of 'stage A' Blaster infection attempts from Aug. 11th 12:00 to Aug. 12th 18:00 .... 66
5.3 Blaster outbreak: Number of 'stage B' Blaster infection attempts from Aug. 11th 12:00 to Aug. 12th 18:00 .... 66
5.4 Blaster outbreak: Number of 'stage C' Blaster infection attempts from Aug. 11th 12:00 to Aug. 12th 18:00 .... 66
5.5 Number of 'stage A' Blaster infection attempts from Aug. 10th to Aug. 15th ........................................ 67
5.6 Number of 'stage B' Blaster infection attempts from Aug. 10th to Aug. 15th ........................................ 67
5.7 Number of 'stage C' Blaster infection attempts from Aug. 10th to Aug. 15th ........................................ 67
5.8 Number of 'stage D' Blaster infection attempts from Aug. 10th to Aug. 15th ........................................ 68
xvi List of Figures

5.9 Number of 'stage E' Blaster infections from Aug. 10th to Aug. 15th .................................. 68
5.10 Blaster worm: ICMP packets per hour .......................................................... 72
5.11 Blaster worm: Unique source addresses per hour ............................................. 73
5.12 Blaster worm: Unique source addresses per 5 minutes ..................................... 73
5.13 Blaster worm: Flows to 135/TCP per hour ...................................................... 74
5.14 Blaster worm: Flows to 135/TCP per 5 minutes ................................................ 74
5.15 Blaster worm outbreak: Flows to 135/TCP per 5 minutes from Aug. 11th 12:00 to Aug. 12th 18:00 ............................................................. 74
5.16 Blaster worm: Flows to 4444/TCP per hour .................................................... 76
5.17 Blaster worm: Bytes to 69/UDP per hour ....................................................... 78
5.18 Blaster worm: Bytes from 69/UDP per hour ................................................... 78
5.19 Blaster worm: Unique source addresses (dest. 69/UDP) per hour ....................... 78
5.20 Sobig.F worm: SMTP traffic volume per hour ................................................ 80
5.21 Sobig.F worm: SMTP flows per hour ............................................................. 80
5.22 Testbed for Sobig.F ......................................................................................... 81
5.23 SMTP flow size distribution before and during Sobig.F .................................... 82
5.24 Number of Sobig.F transmissions over time .................................................... 82

6.1 UPFrame architecture ......................................................................................... 90
6.2 UPFrame buffer handling .................................................................................. 91
6.3 UPFrame traffic shaping on bursty router NetFlow data exports ......................... 93
6.4 The IP activity plot shows stripes of neighbouring IP addresses .......................... 98
6.5 TCP inbound traffic in AS559 for a normal day ............................................... 99
6.6 Blaster and Nachi worm outbreaks .................................................................. 100
6.7 Normal vs. Nachi ICMP incoming network activity ........................................... 101
6.8 The Witty outbreak clearly changes the compressibility ratio ............................ 103

7.1 Host Behaviour Classes .................................................................................. 107
7.2 Sobig.F caused an almost fivefold increase of e-mail traffic volume per hour .... 111
7.3 Cardinalities of the “Traffic ∩ Connector” class two weeks before and during Sobig.F ........................................ 112
7.4 Cardinalities of the “Connector ∩ Responder” class two weeks before and during Sobig.F ........................................ 113
7.5 MyDoom.A: E-mail traffic volume ........................................ 114
7.6 Cardinality of the $T \cap C$ and $T \cap C \cap R$ classes before and during MyDoom.A ........................................ 115
7.7 MyDoom.A worm: SMTP flows ........................................ 116
7.8 MyDoom.A worm: Unique SMTP senders ........................................ 117
7.9 “Connector” class cardinality before and during Witty ........................................ 119
7.10 “Responder” class cardinality before and during Blaster ........................................ 120

9.1 A generic DDoS reflector attack setup ........................................ 128
9.2 Router extension with traffic processing device (TPD) ........................................ 134
9.3 Network model ........................................ 139
9.4 Message sequence diagrams ........................................ 141
9.5 Traffic control unit (TCU) architecture ........................................ 142
9.6 Traffic control unit (TCU) placement for DoS attack mitigation ........................................ 144
9.7 A DDoS reflector attack hits Xorus’ online shop ........................................ 151
9.8 DDoS reflector attack mitigation at the source using the traffic control system ........................................ 151
9.9 Service mapping ........................................ 154
List of Tables

2.1 Backbone and web service provider scenarios . . . . . . . 23
2.2 National economic damage scenarios for Switzerland . . 29
2.3 Estimated Swiss annual economic damage caused by spam 33

5.1 Flows required for infection stages A - E . . . . . . . . 63

7.1 Threshold conditions for host behaviour core classes . . 108

A.1 NetFlow version 5 header format . . . . . . . . . . . 168
A.2 NetFlow version 5 record format . . . . . . . . . . . 169
Chapter 1

Introduction

Internet network security is in its infancy. The unprecedented growth of the Internet, which is currently approaching one billion users worldwide [37], is challenging the network operators. However, security seems still to be commonly regarded as the network users’ duty while network operators primarily focus on fast and reliable data transport around the globe. Most existing security solutions are applicable only to end systems or enterprise networks. Within the Internet backbone, no comprehensive and efficient attack detection and mitigation mechanisms exist and only little is publicly known about the real technical and economical impacts of major recent network security incidents. Four core problems regarding large-scale Internet attacks are:

- Companies and Governments are not aware of the financial damage they could suffer by Internet attacks as there is a lack of a transparent and versatile economic damage model for Internet attacks. This is also a consequence of an evident lack of economic-technical interdisciplinary research and the hesitation to consider intangible damage.

- Only very little is known about how real worms spread in the Inter-
net due to the inaccessibility of detailed long-term Internet traffic traces. Mostly partial or indirect characteristics (e.g. backscatter effects) about real massive Internet attacks are publicly known and a lot of worm research still focuses on theoretical worm spreading simulation models.

- Due to a lack of comprehensive Internet backbone online monitoring systems that can early detect large-scale attacks, reaction to new attacks by network operators is slow, cumbersome and possibly counterproductive.

- Effective countermeasures against large-scale distributed Internet attacks like DDoS\(^1\) currently do not exist. This is mostly due to the open nature of the Internet and the many administrative and technical burdens to counteract an attack originating from worldwide scattered sources spanning multiple network operators’ domains. Whereas the attacker can quite easily grab poorly secured Internet hosts, defence can almost exclusively be done at the victim’s Internet uplink, often far away from the actual attack sources.

This thesis tackles these problems in the following way: Performing an economic analysis of large-scale Internet attacks we came to the conclusion that many companies and economies that rely on the Internet are only vaguely aware of the possibly huge impact of losing Internet connectivity for longer periods. In a second step, we designed algorithms and tools that can help to early detect new massive attacks directly in the Internet backbone in order to counteract faster and more effectively. Finally, we designed a traffic control system based on our fundamental concept of “traffic ownership” that has the potential to give new powerful means to network operators and network users regarding network security and Internet traffic control.

This thesis is organised in three parts that address different aspects of large-scale Internet attacks: In part I) “Impact Analysis”, we look at economic aspects of attacks. In part II) “Early Detection”, we investigate real Internet traffic data and propose a system and algorithms for timely detection of massive attacks before they unfold their full power. Finally, in part III) “Mitigation”, we propose concepts and a system for effectively countering attacks and mitigating their impact. The rest of

\(^1\)DDoS stands for Distributed Denial of Service.
1.1 Impact Analysis

Companies that rely on the Internet for their daily business are challenged by uncontrolled massive worm spreading and the lurking threat of large-scale distributed denial of service attacks. We present a new model and methodology, which allows a company to qualitatively and quantitatively estimate possible financial losses due to partial or complete interruption of Internet connectivity. Our systems engineering approach is based on an analysis of the Internet dependence of different types of enterprises and on interviews with Swiss telcos, backbone and Internet service providers. A discussion of sample scenarios for some companies and for Switzerland illustrates the flexibility and applicability of our model. As an excursus, we also estimate damage caused by spam to the Swiss economy.

The research goal was to show a clear economical need for “security within the network” and to illustrate the potential to prevent huge financial losses if early warning systems for large-scale Internet attacks were in place. In addition, we hope to raise to awareness of how strongly certain companies and the Swiss economy depend on the Internet. The impact analysis also motivates the work done in the other two parts. For this economic damage model [47], we received a best paper award from IEEE.

What is new? The proposed loss model for large-scale Internet security incidents extends existing models by additional important cost factors. Its loss scenarios are the first public ones to transparently illustrate the possible impact of DDoS Internet attacks on Switzerland’s economy.

1.2 Early Detection

The Concise Oxford English Dictionary in its Revised Tenth Edition defines a worm to be “a self-replicating program able to propagate itself across a network, typically having a detrimental effect.” In this second part of the thesis, we discuss current approaches to worm detection,
present our analyses of traffic caused by real worms while spreading through the Internet. We continue by presenting our real-time framework and our algorithms designed for early detection of massive worm spreading events.

1.2.1 Introduction to Attack Detection

We discuss the benefits of early detection, list the data sources used by current detection algorithms, discuss common detection approaches and position our own work within the field of attack detection.

1.2.2 The DDoSVax Project

For our worm analyses and early detection algorithms, we used real world flow-level backbone border traffic data that were gathered in the context of the DDoSVax [134] project at ETH Zürich as a long-term effort. This chapter describes the Internet backbone AS559 operated by SWITCH, where DDoSVax captures traffic data from. It highlights the DDoSVax infrastructure, and introduces flow-level traffic data using Cisco's NetFlow version 5 format.

The aim of DDoSVax was to get access to a large archive of real backbone traffic traces, which could then be used for security and network research. It has proven to be a great advantage to have our own data as getting access to or sharing a large amount of productive network data is legally, organizationally and technically very challenging. The comprehensive flow-level data archive of DDoSVax is worldwide one of the very few ones that are accessible for research purposes.

1.2.3 Flow-Level Worm Traffic Analysis

We present an extensive flow-level traffic analysis of the network worm Blaster.A and of the e-mail worm Sobig.F. Based on packet-level measurements with these worms in a testbed we defined flow-level filters. We then extracted the flows that carried malicious worm traffic from AS559 (SWITCH) border router backbone traffic that were captured in the DDoSVax project. We discuss characteristics and anomalies detected during the outbreak phases, and present an in-depth analysis of partially and completely successful Blaster infections. Detailed flow-level
traffic plots of the outbreaks are given. We found a short network test of a Blaster pre-release, significant changes of various traffic parameters, backscatter effects due to non-existent hosts, ineffectiveness of certain temporary port blocking countermeasures, and a surprisingly low frequency of successful worm code transmissions over the border of backbone AS559 due to Blaster’s multi stage nature and scanning strategy. Finally, we detected many TCP packet retransmissions due to Sobig.F’s far too greedy spreading algorithm.

The research goal was the detailed description of two massive real past Internet attacks (i.e. worm outbreaks) in the way of thorough forensic traffic analyses and herewith revealing new insights into how such events actually look in the backbone.

What is new? Detailed time-line statistics of prevalent worm traffic characteristics like increase in number of scanning hosts, number of successfully infected hosts or backscatter traffic (ICMP). No other similarly detailed results existed for the analysed worms. Most surprising was the low rate of actual worm code transfers (Blaster worm) and the effects of TCP retransmissions due to a greedy spreading behaviour (especially with the e-mail worm Sobig.F) that make security analysis on the flow-level in the backbone harder than expected. Also an important time skew between AS559 external and AS559 internal outbreaks for network worms was detected that underlines the importance and huge loss reduction potential of efficient and early deployed countermeasures.

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

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, e.g. 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 DDoS Vax project. This framework with a custom plug-in is in use 24/7 by ETH Zurich network services and continuously tracks P2P heavy hitters since early 2005. The framework is efficient and robust and can complement traditional intrusion detection systems.

The research goal was the development of an efficient and robust open source framework to de-burst and process flow-level information exported from Internet routers in real-time. It has support for multiple concurrent plug-ins to evaluate different algorithms on the same input and provides the basis for an early-warning system.

What is new? The developed framework is well suited for research and productive use on flow-level data. Memory buffering was optimised for bursty incoming data and low resource demand. We developed our own traffic-shaping mechanism to pace the processing of input data by the plug-ins.

1.2.5 Host Behaviour Based Early Detection of Worm Outbreaks in Internet Backbones

We present a novel near real-time method for early detection of worm outbreaks in high-speed Internet backbones. Our method attributes several behavioural properties to individual hosts like ratio of outgoing to incoming traffic, responsiveness and number of connections. These properties are used to group hosts into distinct behaviour classes. We used flow-level (Cisco NetFlow) information exported by the border routers of a Swiss Internet backbone provider (AS559) that were captured in the DDoS Vax project. By tracking the cardinality of each class over time and alarming on fast increases and other significant changes, we can early and reliably detect worm outbreaks. We successfully validated our method with archived flow-level traces of recent major e-mail based Internet worms such as MyDoom.A and Sobig.F, and fast spreading network worms like Witty and Blaster. Our method is generic in the sense that it does not require any previous knowledge about the exploits and scanning method used by the worms. It can give a set of suspicious hosts in near real-time that have recently and drastically changed their network behaviour and hence are highly likely to be infected.

The research goal was to develop a new generic real-time algorithm that
1.3 Mitigation

Frequency and intensity of Internet attacks are rising at an alarming pace. Several technologies and concepts were proposed for fighting distributed denial of service (DDoS) attacks: traceback, pushback, i3, SOS and Mayday. This chapter shows that in the case of DDoS reflector attacks they are either ineffective or even counterproductive. We then propose our novel and fundamental concept of “traffic ownership” and describe a system that extends the control over network traffic by network users to the Internet using adaptive traffic processing devices. We safely delegate partial network management capabilities from network operators to network users. All network packets with a source or destination address “owned” by a network user can now also be controlled within the Internet instead of only at the network user’s Internet uplink. By limiting the traffic control features and by restricting the realm of control to the “owner” of the traffic, we can rule out misuse of this system. Applications of our system are manifold: prevention of source address spoofing, DDoS attack mitigation, distributed firewall-like filtering, new ways of collecting traffic statistics, service level agreement validation, traceback, distributed network debugging, support for forensic analyses and many more. A use case illustrates how our system enables network users to prevent and to react to DDoS attacks.

The research goal was to propose a novel system that can efficiently mitigate DDoS reflector attacks without the danger of being misused for malicious purposes. A major design goal was flexibility to also support other network based services.

What is new? We invented the fundamental concept of “traffic owner-
ship", which is very powerful to rule out misuse of a distributed Internet traffic control system. We proposed a secure distributed traffic control platform that can be used to effectively counteract Internet attacks and to enable many other interesting new global network based services.

1.4 Research Contributions

This thesis contains five main scientific or engineering contributions to the research in the field of large-scale Internet attacks. They were also published in our peer reviewed conference, workshop and journal papers\footnote{Our papers [45] and [47] received a best paper award from IEEE.} [44], [45], [46], [47], [48], [22] and [24].

- **Economic damage model for DDoS attacks**: Our economic damage model [47] is the first public comprehensive such model that transparently estimates financial damage caused by DDoS attacks suffered by Internet-dependent organisations and entire nations. It reflects experiences of interviewed Internet-dependent companies. Its applicability and usefulness were assured by calculating concrete sample scenarios and by issuing an ETH press statement [51] and presenting it in talks to industry [41, 42], a large insurance company [137], and on an international workshop [47]. The model was well received by the Swiss press and the various international audiences.

- **Blaster worm outbreak analyses**: Our in-depth analyses of the Blaster outbreak [46] are worldwide unique in its level of detail. We were the first to show that mere ingress port blocking on backbone routers during the Blaster outbreak was actually mostly ineffective in preventing new Blaster infections. We also found that worm code transmissions over the backbone border routers were in fact surprisingly rare despite the huge worm scan traffic observed worldwide.

- **Framework “UPFrame” for online algorithm research**: Our light-weight and reliable framework “UPFrame” [48] enables efficient development and testing of near real-time online algorithms on, e.g., NetFlow traces. It was frequently used in the DDoSVax
project and in almost a dozen student theses. An instance of it is used for tracking P2P heavy hitters and runs stable 24/7 since early 2005 at ETH Zurich.

- **Online host behaviour based worm outbreak detection:** Our online worm outbreak detection algorithm [45] is based on a novel classification of the behaviour of Internet hosts. The three behavioural feature classes were designed to be sensitive to worm outbreak events. Using backbone flow-level traffic traces of past network and e-mail based worms it was shown that worm outbreaks are highly visible by tracking the class cardinalities whereas normal traffic goes by unnoticed as desired. This algorithm was implemented as a plug-in for UPFrame and is well suited as a component of an early warning system for massive worm outbreaks.

- **A fundamental concept for a safe distributed traffic control system:** Ideas to incorporate more flexibility into the Internet core like active or adaptive networking were around for many years. However, the hesitation was large by network operators to incorporate any such technologies as data privacy issues and the danger of losing control over the network were not adequately addressed. We proposed a fundamental concept of “traffic ownership” [43, 44, 24], which has the potential to diminish if not eliminate such concern. We also designed a novel traffic control system that incorporates our fundamental concept and enables many previously unthinkable new network-integrated services [44, 24, 22].
Part I

Impact Analysis
Chapter 2

An Economic Damage Model for Large-Scale Internet Attacks

2.1 Introduction

Reliability and availability of Internet services can be degraded significantly within minutes. This became apparent during the massive worm spreading incidents encountered in 2003, namely SQL Slammer [29], Blaster [30], and Sobig.F [31] as well as the distributed denial of service (DDoS) attack on the root domain name servers [76] in October 2002. These worms had a negative impact on the Internet primarily due to their fast and aggressive spreading behaviour and not because of malicious code that could be used to, e.g., launch a massive DDoS attack. We can only conjecture the impact of destructive and well engineered worms on Internet service quality. Flooding of critical connections and vital services with attack traffic could result in total loss of Internet connectivity.
Companies relying on the Internet for their daily business will inevitably sustain substantial financial damage by such a large-scale attack. On the one hand, direct damage (such as e.g. revenue loss during the attack), on the other hand indirect damage (such as e.g. customer loss due to degraded reputation) will be suffered.

For estimating the potential damage when hit by a large-scale Internet attack and for optimising investments into safeguards and preventive security measures, and to assess a “cyber risk” insurance policy, a transparent and comprehensive model and methodology for qualitatively and quantitatively estimating an enterprise’s financial loss caused by massive Internet attacks is indispensable.

2.1.1 Outline

The outline of this chapter is as follows: In the rest of Section 2.1 (Introduction) we present related work, give our system model of the Internet infrastructure together with our threat scenario and the possible user impact. Section 2.2 (Methodology) introduces qualitative damage versus time diagrams and defines the types of damage we consider. We then explain our formulae for loss calculations in Section 2.3 (Calculating Financial Loss). For illustrating the applicability of our method, we give sample scenarios in Section 2.4 (Sample Scenarios) with quantitative loss estimates for companies that rely on the Internet and for the Swiss economy. In Section 2.5 (Estimating Damage by Spam), we present our calculations for economic damage caused by spam. In Section 2.6 (Internet as a Critical Infrastructure), we emphasise that the Internet is becoming a critical infrastructure for modern economies. Finally, we give conclusions and an outlook on possible extensions to our model.

2.1.2 Related Work

Various general risk assessment frameworks such as CMU’s OCTAVE [7] or NIST’s “Federal IT Security Assessment Framework” [97] exist. They help to define a risk management strategy based on a security policy and can be used to identify vulnerabilities and valuable assets. However, their versatility is also their biggest drawback if applied to an Internet interruption attack scenario. Such frameworks give only general guidelines without concrete loss calculations and no explicit system model.
There are other published estimates of DDoS damage, especially for the United States [39]. It is very hard to evaluate their merit since the methodology usually was not published. D.A. Patterson gives in [102] a simple model for roughly estimating downtime cost including degraded employee productivity.

An overview of Internet risk insurance coverage is given in [67]. The calculation models behind such insurances are typically proprietary and confidential, which makes them inaccessible for use in public research. The London (UK) based company mi2g [41] gives comprehensive estimates of economic damage by applying their proprietary and unpublished Economic Valuation Engine for Damage Analysis (EVEDA) algorithm to input from a large number of company managers, hacker bulletin boards, hacker activity monitoring, and anonymous communications with many black hat hackers. They take data attacks (violation of confidentiality, integrity, authentication or non-repudiation of data) and command and control attacks (compromise of computer systems or network equipment) into consideration. With EVEDA, they calculate economic damage caused by the following cyber crimes: covert attacks, spam, phishing, distributed denial-of-service (DDoS), major viruses/worms, and overt attacks. Their estimate for worldwide damage by DDoS attacks in 2004 was 38 bill. US dollars [87]. Total damage by all above mentioned cyber crimes was estimated by mi2g for 2004 to be around 507 bill. US dollars. The order of magnitude of mi2g’s DDoS damage supports the estimates by our economic damage model. As their algorithm is not public, we could not compare it more closely to our approach.

### 2.1.3 Approach

To assure that our model is built in a systematic and scientific way, we based it on the principles of systems engineering [141]. After a conceptual system analysis of the current Internet infrastructure, we categorised loss into various types. Then, we identified and specified relevant elements and their dependences. Finally, in a conceptual synthesis, qualitative and quantitative sample scenarios were established and validated against our model. A well-received industry seminar about our loss model for the large Swiss re-insurance company Swiss Re [137] helped to cross-check our model and sample scenarios.
2.1.4 Internet System Model

Our system analysis resulted in the “Internet system model” depicted in Figure 2.1. It explicits the relevant system elements and their relationships. The figure gives elements in and out of scope of our model. In order to deal with the complexity of today’s Internet infrastructure, it had to be abstracted to a conceptual level. This allowed us to investigate the dependencies that are relevant for estimating direct and indirect financial damage caused by attacks on that infrastructure and sustained by the individual elements.

Core elements of the Internet infrastructure are vital services (e.g. Domain Name Service DNS) and networking devices, mainly routers. A national Internet backbone service provider (BSP) typically hosts only large corporate customers and directly connects them to its high-bandwidth communication infrastructure. Usually, a BSP has several peering points to other national and international BSPs and to smaller Internet service providers (ISPs). Small and medium-sized corporate customers as well as private customers are commonly connected to one of these ISPs, which
are themselves redundantly interconnected to other ISP peers. Many TV
cable network companies also provide Internet connectivity and some of
them also offer voice over IP (VoIP) services to their customers. In
these cases, they also rely on a BSP connection. Traditional telcos typically
have their own voice and data communication infrastructure. Their
network management system (NMS) is commonly based on X.25 or the
Internet Protocol (IP) and runs on a dedicated management network. In
some cases gateways from the NMS to the Internet exist, however today
a telephone company is typically not directly affected by a disruption of
Internet connectivity.

2.1.5 Threat Model and User Impact

Anyone of today's almost one billion Internet users worldwide [37] is a
potential attacker. Many poorly secured computers can be misused by
a remote hacker for an attack. In our threat model, we assume that
the attacker directly or indirectly attacks a large national Internet back-
bone service provider with a massive distributed denial of service attack.
Degradation of backbone bandwidth up to the complete interruption of
Internet connectivity is suffered by the BSP's customers. We assume
that the actual attack stops or is stopped after a relatively short time
(hours to days). Due to the various dependences of the elements in our
system model, such an attack results in massive damage suffered by the
BSP and many other elements – from smaller companies up to entire
nations.

2.1.6 Limitations of Our Damage Model

The complexity of all direct and indirect effects of a massive DDoS In-
ternet attack on an economy is huge and currently infeasible to be fully
mathematically modelled. For the estimate of certain indirect negative
impacts potentially caused by an attack such as a change of the value
of stocks or the effects on company reorganisations due to customer loss
after an incident, no appropriate models exist. In addition, the possible
interrelations and temporal interference of all direct and indirect factors
influenced by such an attack are potentially gigantic.

To keep our model simple and to guarantee its applicability for use by
enterprises, we only modeled a small set of clearly defined direct and
indirect damage types (see Section 2.2.2) for the elements specified in our system model (see Section 2.1.4). Furthermore, feedback relations between affected companies that depend on each other to fulfill a service were not taken into account. However, our sample scenarios (see Section 2.4) assured us that our model can be used as a reasonable basis for estimating potential damage caused by a massive attack.

2.2 Methodology

Since the effects of a DDoS attack are complex, we took several steps in order to analyse interdependences involved and financial loss incurred by this type of attack. Specifically, we use graphical plots representing damage versus time in a qualitative fashion, mathematical formulae that can be used to calculate the financial loss for the different types of damage, and example scenarios that demonstrate how to calculate financial loss for concrete settings.

Estimating the probability of different types of DDoS attacks is very hard. Estimating the time needed to stop such an attack, if the attacker has designed the attack system in a way that the attack is robust against basic traffic filtering, is currently pure guesswork. Cleanup durations in the range of several weeks seem possible for DDoS attacks launched by sophisticated polymorphic malicious Internet worms.

The main aim of our model and methodology is to provide a universal tool that can be used to calculate the expected financial loss for a variety of cyber attack scenarios involving Internet DDoS attacks. We focus only on the economic damage and will not elaborate on when to transfer the risk of being hit by a cyber attack by getting insurance.

2.2.1 Damage vs. Time

Financial loss is an expected effect of any significant degradation of Internet performance. Furthermore financial loss changes over time. Economic damage usually has not the same characteristics over time as technical problems have. Economic damage can still grow when technical problems have been resolved and the attack has stopped. It is therefore reasonable to evaluate damage for the time $t \to \infty$ as a first approach. Several examples can be found in Section 2.4 and in Figures
2.2 Methodology

2.2, 2.3, 2.4, and 2.5. We set $t = t_0$ as the time at which the attack starts and $t = t_1$ as the time at which the attack has been completely stopped. The time span $[t_1, t_2]$ represents time shortly after the attack, while $t > t_2$ refers to time a longer time after the incident. Times $t > t_2$ may e.g. be weeks to months later. The reason why we introduce $t_2$ is that it allows us to separate the damage effects of liability cost and customer loss in the qualitative damage estimates given in Section 2.4. For real attack cases, we expect the cost of the types of damage that arise typically after the attack has been stopped (i.e. in $t > t_1$) in most cases to greatly overlap each other in time. Therefore, instead of setting an exact time point for $t_2$ in an attack case, it is advisable to rather judge when how much of which cost of the types of damage in $t > t_1$ will arise on a case by case basis. In the very improbable case that an attack lasts several weeks or even months, the cost that typically arise after $t_1$ could accumulate even before the attack has been fully stopped. We do not model the case that the technical problems cannot be solved for a longer time or cannot be solved at all.

2.2.2 Types of Damage

We subdivide financial damage (as the result of the interruption of Internet services) into four categories:

**Downtime Loss** The downtime loss can be split further into *productivity loss* (employees can no longer do “business as usual” and have to use less efficient ways to fulfill their duties; certain tasks can only be done later) and *revenue loss* (lost transactions by customers that cannot access a service or due to the inability of a company to fulfill customer requests).

**Disaster Recovery** Costs of the time that employees and external staff have to spend on recovery from an incident. Additionally, material costs can arise.

**Liability** Many companies offer service level agreements (SLAs) to their customers. In case that their service quality deviates from an SLA, the customers can claim compensation payments. Liability related losses can be partially insured and typically arise several days after the incident.
Customer Loss Customers being dissatisfied by degraded service quality might terminate their service contract. The rate of new customers joining a service can substantially drop if the reputation of a company suffers. These opportunity costs arise typically weeks to months after an incident.

2.3 Calculating Financial Loss

Financial loss has to be quantified in monetary units. While the mathematics used is relatively basic, there are many factors to be considered. The factors present in our formulae stem from the synthesis of interviews with Internet dependent enterprises [26, 50, 138]. The calculations are only approximations. However, they can serve as a basis to estimate how much investment into improved infrastructure robustness and faster disaster recovery is justified. The following subsections describe the details for each type of loss. The legend for the symbols used in the formulae can also be found in Table 2.1 together with their units.

2.3.1 Downtime

Total downtime related loss is the sum of productivity and revenue loss. This type of loss is incurred during the actual downtime time span $[t_0, t_1]$. We get

$$L_D = \frac{E_{ca}}{d_a} \cdot d_o \cdot E_{no} \cdot E_{po} + \frac{R}{d_{so}} \cdot d_{so} \cdot R_o \cdot S_o$$

$L_D$: Loss due to downtime costs
$E_{ca}$: Annual cost per employee
$d_a$: Working hours per employee and year
$d_o$: Working hours overlapping outage time
$d_{so}$: Service operating hours per year
$d_{so}$: Service operating time affected by outage
$E_{no}$: Employees affected by outage
$E_{po}$: Productivity degradation during outage (0-100%)
$R_a$: Total annual revenue
$R_o$: Part of the revenue affected by a full outage not realised later (0-100%)
$S_o$: Degree of service degradation (0-100%)
2.3 Calculating Financial Loss

2.3.2 Disaster Recovery

The loss due to disaster recovery is the sum of the cost for work and material to get the system up and running again. It arises during the downtime time span \([t_0, t_1]\). We get

\[ L_r = E_r \cdot E_{ch} \cdot d_r + M_c \]

- \(L_r\): Loss due to recovery costs
- \(E_r\): Number of recovery team members
- \(E_{ch}\): Hourly cost for a recovery team member
- \(d_r\): Recovery work outside office hours
- \(M_c\): Cost of material needed

2.3.3 Liability

This loss class describes cost incurred because contracts with third parties cannot be fulfilled and consequently these third parties demand financial compensation. The loss is incurred during time span \([t_1, \infty]\) and equals the sum of all demands:

\[ L_C = \sum C_c + \sum C_l \]

- \(L_C\): Loss due to liability claims
- \(C_c\): Claims from contractual penalties
- \(C_l\): Claims from other liabilities

If a claim is in dispute, substantial legal costs may arise in addition. This type of loss can often only be quantified when the affected third parties make their claims known. It is hard to estimate how much an affected third party was actually damaged by the outage without asking it. ISPs typically reimburse their customers for the time they were unable to provide service.

2.3.4 Customer Loss

If a service is unavailable for some time, customers might move to another service provider or no longer use the service. This type of loss is incurred
over a very long time span of up to \([t_2, \infty]\) and also includes loss of potential new customers. We get

\[
L_{CL}(\Delta t) = [C_A(\Delta t) + C_P(\Delta t)] \cdot R_C(\Delta t)
\]

\[
\begin{align*}
L_{CL} & : \text{Financial loss due to customer loss} \\
\Delta t & : \text{Time span } [t_2, t_x], \text{ with } t_2 < t_x < \infty \\
C_A & : \text{Number of actual customers lost in } \Delta t \\
C_P & : \text{Number of potential customers lost in } \Delta t \\
R_C & : \text{Average revenue per customer}
\end{align*}
\]

If the revenue per customer varies significantly, the above expression is inaccurate and should be replaced by a detailed analysis focused on the most important customers.

### 2.4 Sample Scenarios

In the following, we discuss some sample scenarios that illustrate the loss expectancy for different types of enterprises. For each scenario we plot the qualitative cumulative financial loss over time.

In order to demonstrate the flexibility and applicability of our model and methodology, Tables 2.1 and 2.2 show sample calculations that quantify the economic damage for four concrete scenarios. Monetary unit is Swiss Francs (CHF). In August 2005, the currency conversion was 1.00 CHF = 0.64 EUR = 0.79 USD.

#### 2.4.1 Estimate Precision

Where we had statistical sources for estimating parameters for our sample scenarios, we referenced them in a footnote of the corresponding table. However, there were still several parameters involved like productivity degradation, number of affected employees, part of revenue affected, recovery hours required, liability claims or customers lost that had to be guessed without having any reliable statistics or data sources. Nevertheless, we calculate the damage cost in the unit of single Swiss Francs. The reason is to make it easier to follow our calculations. However, the actual precision of our estimates is much lower. As there are no comparable such damage estimates or measurements available, we
### 2.4 Sample Scenarios

#### Table 2.1: Backbone and web service provider scenarios

<table>
<thead>
<tr>
<th>Factor</th>
<th>Symbol</th>
<th>Unit</th>
<th>BSP</th>
<th>WSP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outage parameters</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Outage time</td>
<td>$d_o$</td>
<td>h</td>
<td>24</td>
<td>168</td>
</tr>
<tr>
<td>Working hours overlapping outage time</td>
<td>$d_a$</td>
<td>h</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>Service operating time affected by outage</td>
<td>$d_{so}$</td>
<td>h</td>
<td>24</td>
<td>168</td>
</tr>
<tr>
<td>Degree of service degradation</td>
<td>$S_o$</td>
<td></td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Downtime Loss</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Degraded Productivity</td>
<td>$E_{ca}$</td>
<td>CHF/yr</td>
<td>98,075</td>
<td>98,075</td>
</tr>
<tr>
<td>Working time per employee and year</td>
<td>$d_e$</td>
<td>h/yr</td>
<td>1,880</td>
<td>1,880</td>
</tr>
<tr>
<td>Employees affected by outage</td>
<td>$E_{na}$</td>
<td></td>
<td>3,500</td>
<td>4</td>
</tr>
<tr>
<td>Productivity degradation during outage</td>
<td>$E_{po}$</td>
<td></td>
<td>20%</td>
<td>20%</td>
</tr>
<tr>
<td><strong>SUM</strong></td>
<td>CHF</td>
<td>292,128</td>
<td>1,669</td>
<td></td>
</tr>
<tr>
<td>Loss of Revenue</td>
<td>$R_o$</td>
<td>CHF/yr</td>
<td>2,815 mill.</td>
<td>1,000,000</td>
</tr>
<tr>
<td>Service operating hours per year</td>
<td>$d_{so}$</td>
<td>h</td>
<td>8,760</td>
<td>8,760</td>
</tr>
<tr>
<td>Part of the revenue affected by full outage</td>
<td>$R_o$</td>
<td></td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td><strong>SUM</strong></td>
<td>CHF</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>SUM for Downtime</strong></td>
<td>$L_D$</td>
<td>CHF</td>
<td>292,128</td>
<td>1,669</td>
</tr>
<tr>
<td><strong>Disaster Recovery</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of recovery team members</td>
<td>$E_r$</td>
<td></td>
<td>1,750 (50%)</td>
<td>0</td>
</tr>
<tr>
<td>Hourly cost for a recovery team member</td>
<td>$E_{ch}$</td>
<td>CHF</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>Recovery work outside office hours</td>
<td>$d_r$</td>
<td>h</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>Cost of material needed</td>
<td>$M_c$</td>
<td>CHF</td>
<td>1,000,000</td>
<td>0</td>
</tr>
<tr>
<td><strong>SUM for Recovery</strong></td>
<td>$L_r$</td>
<td>CHF</td>
<td>5,200,000</td>
<td>0</td>
</tr>
<tr>
<td><strong>Liability</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Claims from contractual penalties</td>
<td>$C_c$</td>
<td>CHF</td>
<td>15,000,000</td>
<td>0</td>
</tr>
<tr>
<td>Claims from other liabilities</td>
<td>$C_l$</td>
<td>CHF</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>SUM for Liability</strong></td>
<td>$L_{CL}$</td>
<td>CHF</td>
<td>15,000,000</td>
<td>0</td>
</tr>
<tr>
<td><strong>Customer Loss</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time span</td>
<td>$\Delta t$</td>
<td>yrs</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Number of actual customers lost</td>
<td>$C_A$</td>
<td>CHF/yr</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>Number of potential customers lost</td>
<td>$C_P$</td>
<td></td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>Average revenue per customer</td>
<td>$R_C$</td>
<td></td>
<td>500,000</td>
<td>1,300</td>
</tr>
<tr>
<td><strong>SUM for Customer Loss</strong></td>
<td>$L_{CL}$</td>
<td>CHF</td>
<td>12,500,000</td>
<td>169,000</td>
</tr>
<tr>
<td><strong>Total Economic Loss (ca. ±30%)</strong></td>
<td>CHF</td>
<td>32.99 mill.</td>
<td>0.17 mill.</td>
<td></td>
</tr>
</tbody>
</table>

Comments (for table rows indicated):
- 0 BSP: Backbone Service Provider; WSP: Web Service Provider
- 1 Source: Swiss federal office for statistics [19, 20]
- 2 40 hours week and 5 weeks of vacations per year
- 4 Computer based work limited, no e-mail, no Internet
- 6 WSP: 6 employees, 800 customers, 2500 domains, CHF 1 mill. annual revenue
- 9 WSP: Assumes a flat rate for the data volume
- 10 WSP: Recovery is not a responsibility of the WSP
cannot derive a statistical distribution and give a confidence interval for our own estimates. For the web service provider and backbone service provider sample scenarios given in Table 2.1, we think that the estimate precision is around ±30% of the given total damage. For the Swiss national scenario, we judge the precision to be around ±20% of the given total damage. The averaging over all IT-dependent businesses makes our Swiss national damage estimates more exact than our single company damage estimates.

Significant improvement of the estimate precision is possible for an Internet dependent company, where company confidential information about business processes, staff salaries, revenue per customer, service contracts and service disruption penalties are accessible. Concretising an attack and knowing a company’s infrastructure topology and dependence can further improve estimate precision. Possibly, the employees are already experienced from past (minor) incidents that typically were not made public. In addition, the average salaries we used could be replaced with salaries directly derived from the usually confidential personnel cost of employees and external service partners.

2.4.2 Backbone and Internet Service Providers

![Diagram of Total Financial Loss over time](image)

**Figure 2.2:** Loss of a BSP, an ISP and a web service provider

During the downtime \([t_0, t_1]\) employee productivity is low as Internet related services such as e-mail and web based communication are no longer available. Branch offices connected through virtual private networks (VPNs) are disconnected. In case that a BSP or ISP offers hosting or interconnection with pricing based on data transfer volume or if revenue is earned by showing ads on, e.g., a portal web site, financial loss corresponding to the service fees lost will be suffered. Having customers
2.4 Sample Scenarios

paying a flat fee is advantageous in this event. Productivity and revenue loss sum up to the *downtime loss*, which grows linearly with the length of the downtime. *Disaster recovery* mainly consists of additional work hours of network operators and grows linearly as well.

BSPs are hit stronger by *liability* claims than ISPs as unsatisfied customers of a BSP can often refer to an SLA and claim compensation. Best-effort guarantees common for ISPs help to reduce such claims. However, a partial reimbursement of paid flat fees might occur.

As unhappy customers cannot immediately cancel a contract, the damage resulting from *customer loss* might occur weeks or even months after the actual technical incident. A sudden surge of customers terminating their contracts is likely to happen at the end of the current service period.

The backbone service provider (BSP) scenario in Table 2.1 assumes complete interruption of Internet dependent services for 24 hours. The figures are chosen such as to match a BSP of the size of Swisscom Fixnet Wholesale (SFW). Productivity degradation is estimated at 20% as most employees can do other pending work unrelated to the Internet. Total estimated loss is about CHF 33 mill., which corresponds to 1.2% of SFW’s annual income.

2.4.3 Corporate Customers

For large corporate customers of a BSP/ISP the *productivity loss* is similar to the BSP/ISP scenario discussed in Subsection 2.4.2. An e-shop that sells only over the Internet also suffers severe *revenue loss* as e-shop customers that cannot connect to this online shop can easily buy
in another one, which is currently available. Large companies and corporations typically sell over various channels and hence suffer a lot less revenue loss in case of Internet interruption. The resulting downtime cost grows linearly. Disaster recovery costs are rather small as the prevalent technical problems are typically solved by the ISP or BSP. Liability claims are rare to occur for short business interruptions as is shown in the diagram. However, if an e-shop sells strongly time dependent goods on behalf of others and under a service level agreement, e.g., tickets for events, then compensation payments for service unavailability might occur. For long-term interruptions such claims can become a major issue. The same is true regarding customer loss.

2.4.4 Web Service Providers

Web service providers often charge customers for their data transfer volume like ISPs do. The total loss due to downtime, disaster recovery, and liability is analogous in its characteristics to the ISP scenario in Section 2.4.2. For the cumulative loss graph in Figure 2.2, we assumed no liability costs for the web service provider.

The damage due to customer loss depends heavily on the type of hosted customers. Infrequent and short interruptions will rarely be noticed by private customers, whereas e-shops can suffer a significant loss. A worst case would occur, if the web service provider’s servers get broken into due to a lack of security, which would unsettle many customers.

The web service provider (WSP) scenario assumes a one week complete interruption of Internet dependent services. Our sample WSP with 6 employees hosting 2500 domains of 800 customers and having CHF 1 mill. in annual revenue suffers an estimated loss of CHF 0.17 mill. for the assumptions listed in Table 2.1.

2.4.5 Insurance Companies

Use of modern communication technologies such as the Internet to enhance a company’s productivity are inevitable. However, many companies just slowly become aware that their financial success heavily depends on an “always-on” Internet. Traditional insurance policies such as corporate liability policies [15] are not adequate to protect a company
from business interruptions, productivity degradation and financial loss caused by Internet attacks.

The Swiss insurance company Zurich offers an “eRisk protection program” since mid-2000. This service includes consulting for risk analysis, legal advice, and optimisations to Internet related security. Business interruption, data, software, and public relations cost, and also liability claims in case of service interruptions can be covered. The re-insurance company Swiss Re offers consulting and solutions for non-physical damage business interruptions, cyber liability, and revenue protection. However, the details of such risk analyses and policy calculations are proprietary and confidential and hence difficult to compare. Today, most BSPs and ISPs still refrain from obtaining insurance coverage for Internet related cyber risks.

The damage suffered by insurance companies in the event of a large-scale Internet attack is mainly the sum of liability claims from insurance policies. The graph in Figure 2.4 does not show the comparably small productivity loss incurred.

### 2.4.6 Telcos

As telephone networks, which generate the biggest part of the revenue for a telco, are usually separate from the Internet infrastructure, a telco suffers primarily from productivity loss of its employees that can no longer use the Internet during an attack. It is possible that a telco generates additional revenue during an attack due to people calling others by phone instead of sending e-mails.
2.4.7 TV Cable Companies

If a TV cable company only provides television broadcast services, the scenario is comparable to the one for a telco as just described. In case that broadband Internet is offered over the TV cable, the loss characteristics are similar to the ISP scenario discussed in Subsection 2.4.2.

2.4.8 Swiss National Scenarios

Table 2.2 gives two Swiss national scenarios. According to [38] 48.2% of all 3,590,000 employees [20] working in Switzerland do an IT intense job. This results in 1,730,380 employees affected by a massive DDoS attack that causes an Internet blackout and that lasts one week (168 hours). The economic damage of such an event to the Swiss economy with an annual GDP of CHF 482 billion sums up to CHF $\sim$6 billion, i.e. 1.2% of GDP. For an Internet outage of a single working day in our Swiss national scenario, we assumed that only 60% respectively 1,038,228 (i.e. all large enterprises and a part of the SMEs) of all employees in IT intense jobs are affected. In addition, the Swiss national scenarios do not account liability claims and loss of customers since it is assumed that liability is within Switzerland and no customers are lost.
### 2.4 Sample Scenarios

#### Swiss National Scenarios

<table>
<thead>
<tr>
<th>Factor</th>
<th>Symbol</th>
<th>Unit</th>
<th>Swiss National Scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outage parameters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Outage time</td>
<td>$d_o$</td>
<td>h</td>
<td>24</td>
</tr>
<tr>
<td>Working hours overlapping outage time</td>
<td>$d_w$</td>
<td>h</td>
<td>8</td>
</tr>
<tr>
<td>Service operating time affected by outage</td>
<td>$d_{so}$</td>
<td>h</td>
<td>24</td>
</tr>
<tr>
<td>Degree of service degradation</td>
<td>$S_o$</td>
<td></td>
<td>100%</td>
</tr>
</tbody>
</table>

#### Downtime Loss

<table>
<thead>
<tr>
<th></th>
<th>Degraded Productivity</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Annual cost per employee</td>
<td>$E_{c_e}$</td>
<td>CHF/yr</td>
</tr>
<tr>
<td>2</td>
<td>Working time per employee and year</td>
<td>$d_{w}$</td>
<td>h/yr</td>
</tr>
<tr>
<td>3</td>
<td>Employees affected by outage</td>
<td>$E_{n_o}$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Productivity degradation during outage</td>
<td>$E_{p_o}$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>SUM</td>
<td></td>
<td>CHF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Loss of Revenue</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Total annual revenue</td>
<td>$R_e$</td>
<td>CHF/yr</td>
</tr>
<tr>
<td>7</td>
<td>Service operating hours per year</td>
<td>$d_{so}$</td>
<td>h</td>
</tr>
<tr>
<td>8</td>
<td>Part of the revenue affected by full outage</td>
<td>$R_{c_e}$</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>SUM</td>
<td></td>
<td>CHF</td>
</tr>
</tbody>
</table>

#### SUM for Downtime | $L_D$ | CHF | 284,741,095 | 5,502,928,061 |

#### Disaster Recovery

| | Number of employees in the recovery team | $E_r$ | | 10,382 (1%) | 17,304 (1%) |
|---|-----------------------------------------|---|---|
| 11 | Cost per hour for a recovery team member | $E_{c_r}$ | CHF | 150 | 150 |
| 12 | Recovery work hours outside office hours | $d_{r}$ | h | 16 | 128 |
| 13 | Cost of material needed | $M_c$ | CHF | 0 | 0 |
| 14 | SUM for Recovery | $L_{r}$ | CHF | 24,917,472 | 332,232,960 |

#### Liability

| | Claims from contractual penalties | $C_{c}$ | CHF | 0 | 0 |
|---|----------------------------------|---|---|
| 16 | Claims from other liabilities | $C_{l}$ | CHF | 0 | 0 |
| 17 | SUM for Liability | $L_C$ | CHF | 0 | 0 |

#### Customer Loss

<table>
<thead>
<tr>
<th></th>
<th>Time span</th>
<th>$\Delta t$</th>
<th>yrs</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>Number of actual customers lost</td>
<td>$C_{A}$</td>
<td>CHF</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>Number of potential customers lost</td>
<td>$C_{P}$</td>
<td>CHF</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>Average revenue per customer</td>
<td>$R_{c}$</td>
<td>CHF/yr</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>SUM for Customer Loss</td>
<td>$L_{CL}$</td>
<td>CHF</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Total Economic Loss (ca. ±20%) | CHF | 309.66 mill. | 5.84 bill. |

Comments (for table rows indicated):
1. Source: Swiss federal office for statistics [19, 20]
2. 40 hours week and 5 weeks of vacations per year
4. Computer based work limited, no e-mail, no Internet
6. Source: Swiss federal office for statistics [18], Credit Suisse [38]
17. Claims only within Switzerland
22. No customers are lost

Table 2.2: National economic damage scenarios for Switzerland
2.5 Estimating Damage Caused by Spam

Our economic damage model was built on the assumption of an Internet DDoS attack scenario that causes severe degradation of Internet services. However, unwanted costs arise also daily in companies with Internet access due to cyber criminal activities. In this section, we show as an excursus how to calculate damage caused by spam e-mails. We define “spam” as unwanted e-mails containing questionable business offers, unsolicited information, phishing attacks\(^1\) or malcode such as, e.g., virus attachments. According to the spam and virus intercept statistics by Message Labs \([3]\) for June 2005, 67.25\% of all e-mails scanned for their customers were spam and 1 in 28.16 e-mails scanned contained a virus.

We now investigate the costs caused by productivity loss\(^2\) and by taking preventive measures suffered by enterprises that try to minimise the annoying flood of spam reaching employees.

2.5.1 Productivity Loss

Basically, productivity loss arises due to an employee having to daily scan through and delete new spam and virus infected e-mails that were delivered to his or her electronic mail inbox. In addition, he or she might need time to install or update a local spam and virus protection on his or her computer. Visiting educational events and reading corporate security related news regarding new e-mail threats is also accounted to productivity loss caused by spam.

The annual time spent for this activity by all employees of a single company gives the total annual productivity loss \(ProdLoss_{\text{spam}}\), more precisely:

\[
ProdLoss_{\text{spam}} = d_{\text{work}} \cdot \left( \sum_{i=1}^{n_{\text{mail}}} t_{e_i,d,\text{spam}} \cdot c_{e_i,h} \right)
\]

\(^1\) In a phishing attack, the victim is typically asked by e-mail to disclose confidential access codes such as passwords or login names for web services by urging the victim to visit a faked web site layouted in the design of a well-known trusted company.

\(^2\) Productivity loss is part of the downtime loss in our DDoS attack damage model.
2.5 Estimating Damage Caused by Spam

\[ d_{\text{work}} \] : Workdays per year

\[ n_{\text{mail}} \] : Number of employees that use e-mail

\[ t_{e_i, \text{d,spam}} \] : Average time per day (in hours) spent on spam by employee \( e_i \)

\[ c_{e_i,h} \] : Average hourly cost of employee \( e_i \)

We used this formula and some averaging to estimate the productivity loss by all employees having IT intense jobs in Switzerland. Assuming an average daily time of 5 minutes wasted on “spam” (as defined above) per employee, we get an annual economic damage due to productivity loss of approximately CHF 1.7 billion. The details of our calculations can be found in Table 2.3.

2.5.2 Preventive Measures

Enterprises using the Internet try to cut down productivity loss caused by spam. They use e-mail filtering software to detect and mark spam e-mails and to delete e-mails with malcode in attachments. However, such filters are far from perfect and need to be updated and adjusted when new forms of spam arrive.

Diverse activities are needed for proper spam prevention: anti-spam system installation and maintenance, user support, reading security news, education of administrators, security awareness events for users, distribution of spam system updates, etc. These activities result in personnel cost and licence fees, which also contribute to the economic damage of spam.

For estimating cost of preventive measures we treated enterprises differently depending on whether they were large, small and medium (SME) or micro enterprises. Whereas large enterprises presumably all have e-mail, a security sysadmin and can afford full featured anti-spam solutions, we assume that only about 80% of SMEs use e-mail and buy smaller anti-spam systems, and that merely 50% of micro enterprises have e-mail and that their employees have to fight spam without a dedicated security sysadmin helping them. Our calculations, as presented in Table 2.3, result in annual personnel cost for fighting spam of CHF \(~480\) mill., and annual anti-spam licence cost of \(~24\) mill., which result in total annual spam prevention cost for enterprises in Switzerland of CHF \(~504\) million.
2.5.3 Discussion

Adding the productivity loss of CHF ~1.7 billion to the preventive cost of CHF ~0.5 billion, we get a total annual economic damage for Switzerland caused by spam of CHF ~2.2 billion.

Regarding this huge economic damage for a small country like Switzerland, it astonishes how slow politicians search for legal and organisational means for fighting spam effectively, e.g. by rigorously fining spam senders and virus creators. Without fundamentally changing today’s e-mail system (e.g. requiring sender authentication, charging a fee per e-mail sent etc.), additional means are direly needed to supplement technical anti-spam solutions.
### 2.5 Estimating Damage Caused by Spam

<table>
<thead>
<tr>
<th>Factor</th>
<th>Symbol</th>
<th>Unit</th>
<th>Spam in CH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Productivity degradation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total jobs in CH (100%)</td>
<td>( n_{IT} )</td>
<td></td>
<td>3,590,000.00</td>
</tr>
<tr>
<td>IT intense jobs (48.2%)</td>
<td>( n_{IT} )</td>
<td>CHF/yr</td>
<td>1,730,380.00</td>
</tr>
<tr>
<td>Annual cost per employee</td>
<td>( c_{empl} )</td>
<td>CHF/yr</td>
<td>98,075.00</td>
</tr>
<tr>
<td>Working time per employee and year</td>
<td>( d_{work} )</td>
<td>d/yr</td>
<td>230</td>
</tr>
<tr>
<td>Daily cost per employee</td>
<td>( c_{d} )</td>
<td>CHF/d</td>
<td>426.41</td>
</tr>
<tr>
<td>Hourly cost per employee</td>
<td>( c_{h} )</td>
<td>CHF/h</td>
<td>52.17</td>
</tr>
<tr>
<td>Time per day spent on spam e-mails (5 min/d)</td>
<td>( t_{d,spam} )</td>
<td>h</td>
<td>0.08</td>
</tr>
<tr>
<td>Daily spam cost per employee</td>
<td>( c_{d,spam} )</td>
<td>CHF</td>
<td>4.35</td>
</tr>
<tr>
<td>Total annual productivity loss due to spam</td>
<td>( n_{IT} \cdot c_{d,spam} \cdot d_{work} )</td>
<td>CHF</td>
<td>1.7 bill.</td>
</tr>
</tbody>
</table>

### Spam Prevention Cost

| Statistics about enterprises in CH | | |
|-----------------------------------|----------------|
| Number of enterprises in CH (100%) | 317,739.00 |
| Large enterprises (0.30%) | 953.22 |
| Small and medium enterprises SME (11.7%) | 37,175 |
| Micro enterprises (88%) | 279,610 |

| Personnel cost (for spam prevention) | | |
|-------------------------------------|----------------|
| Annual cost per security sysadmin | \( c_{adm} \) | CHF/yr | 150,000 |
| Annual cost per employee | \( c_{empl} \) | CHF/yr | 98,075.00 |

| Large enterprises | | |
|-------------------|----------------|
| Employment ratio of sysadmin for spam | \( e_{le} \) | 50% |
| Annual cost for “spam” sysadmin per large enterprise | \( f_{le} \) | CHF/yr | 75,000.00 |
| Percentage of large enterprises having e-mail | \( f_{le} \) | 100% |

| SME | | |
|-----|----------------|
| Employment ratio of sysadmin for spam | \( e_{sme} \) | 3% |
| Annual cost for “spam” sysadmin per SME | \( f_{sme} \) | CHF/yr | 4,500 |
| Percentage of SMEs having e-mail | \( f_{sme} \) | 80% |

| Micro enterprises | | |
|------------------|----------------|
| Employment ratio of dedicated employee for spam (~10 min/d) | \( e_{me} \) | 2% |
| Annual cost for “spam” employee per micro enterprise | \( f_{me} \) | CHF/yr | 1,961.5 |
| Percentage of micro enterprises having e-mail | \( f_{me} \) | 50% |

| Total for micro enterprises | | |
|----------------------------|----------------|
| \( n_{sme} \cdot c_{adm} \cdot e_{sme} \cdot f_{sme} \) | CHF | 274,227,821 |

| Total annual personnel cost | \( n_{sme} \cdot c_{empl} \cdot e_{sme} \cdot f_{sme} \) | CHF | 480 mill. |

| Licence cost (for spam prevention) | | |
|------------------------------------|----------------|
| Annual licence cost per virus/spam scan solution | \( l_{sc} \) | CHF/yr | 50.00 |
| Large enterprises: annual cost for central spam solution | \( a_{c} \) | CHF/yr | 10,000 |
| Licence cost for all large enterprises: \( n_{sme} \cdot l_{sc} \) | | CHF | 9,532,170 |
| SME: number of licences | \( l_{sme} \) | 5.00 |
| Licence cost for all SMEs: \( n_{sme} \cdot f_{sme} \cdot l_{sme} \cdot l_{sc} \) | CHF | 7,435,093 |
| Micro enterprises: Number of licences | \( l_{me} \) | 1.00 |
| Licences for all micro enterprises | \( n_{sme} \cdot f_{sme} \cdot l_{sme} \cdot l_{me} \cdot l_{sc} \) | CHF | 6,990,258 |

| Total annual licence cost | | |
|--------------------------|----------------|
| \( n_{sme} \cdot c_{empl} \cdot e_{sme} \cdot f_{sme} \) | CHF | 24 mill. |

| Total annual cost for spam prevention | | |
|--------------------------------------|----------------|
| \( n_{IT} \cdot c_{d,spam} \cdot d_{work} \) | CHF | 504 mill. |
| Total annual spam damage for Switzerland (ca. ±20%) | | CHF | 2.2 bill. |


Table 2.3: Estimated Swiss annual economic damage caused by spam
2.6 Internet as a Critical Infrastructure

Modern economies are unthinkable without efficient and cost-effective worldwide information exchange. The importance of the Internet for international communication is still growing: More and more enterprises start to trade exclusively over the Internet and stop to accept orders by postal mail or telephone. Voice over IP (VoIP) becomes increasingly popular. Enterprises and even network operators plan to quit traditional telephone services and start integrating voice communication into their network. The evident driving force of this integration is cost reduction as a result of sharing the already existing worldwide Internet infrastructure for data and voice communication. Additionally, large network companies like Cisco recently made attractive offers for prospective users in order to push VoIP to the market. Questions like how to bootstrap the Internet after a major breakdown, e.g., due to an exploit of a severe router vulnerability, or how to effectively defend the network during a massive DDoS attack on a backbone without having an alternate communication channel between Internet providers currently die away unheard.

However, these developments increase the dependence of modern economies on the Internet at an alarming pace. We consider this trend of reducing redundancy almost unawarely as dangerous. We think that the Internet must soon be regarded as a critical infrastructure for modern economies similar to energy and water supply systems. The protection of such a system needs definitely more attention than it gets today not only from technical but also from political people.

When discussing threats faced by critical infrastructures, it seems reasonable to extend the definition of risk. According to ISO/IEC [74] “risk” is defined as the combination of the probability of an event and its usually negative consequences to assets. Prof. Kröger, the founding rector of the International Risk Governance Council IRGC [5] extended the definition of risk with regard to critical infrastructures in [79, 80]: “The ’asset’ includes in particular continuity of service; its loss can be judged against its criticality for people, society and sectors for which scope, magnitude and effects of time as well as amplifying factors and secondary effects are taken into account. Risk must be understood as a dynamic concept linked to context and, therefore, subject to continued changes.” Due to the possibility of large-scale DDoS attacks the availability of Internet services clearly is at risk.
2.7 Conclusions and Outlook

The threat potential of massive DDoS attacks on critical Internet infrastructure elements and the damage caused by spam and viruses can no longer be ignored. The possible direct and indirect financial loss for many companies must be considered when operating a strongly Internet dependent business.

Our model and methodology provides a basis for transparent financial damage estimates in a qualitative and a quantitative way. The sample scenarios show that our model and methodology are well applicable to real world situations and that the calculations involved are straightforward. Damage estimates by our model can be used for assessing the vulnerability of Internet dependent companies, for optimising investments into safeguards and preventive security measures, and to assess a "cyber risk" insurance policy.

We propose to consider some possible minor refinements of our calculations: The grade of productivity degradation during outage could be modelled with, e.g., an exponential function that increases the grade of degradation for each additional day of an outage instead of with an average degradation for the whole outage duration. The cost of bad reputation due to service interruption was modelled merely by customer loss, but could additionally incorporate the additional labour and marketing efforts to find and hold new customers in such a situation. The recovery cost could be complemented by the cost for building and maintaining an alternate site in hot or cold stand-by.

As larger extensions to our model we propose to consider preventive measures taken against massive attacks, new kinds of threats that involve different loss characteristics over time or even company reorganisations due to incidents. The underlying Internet system model could be enlarged to encompass aspects of the stock market and other indirectly affected systems.
Part II

Early Detection
Chapter 3

Attack Detection
Approaches

Attacks in the Internet are not a new phenomenon. The first known Internet worm is the “Morris worm” [122] named after its author and dates back to Nov. 2nd, 1988 to the early ages of the Internet. This worm used a software buffer overflow exploit to break into and consequently infect remote machines. The effects were devastating to the early Internet as thousands of machines were infected, and normal activities and Internet connectivity were disrupted for many days. Nowadays, Internet worms and attacks can be seen daily. However, many of them have only minor impacts on the network and are therefore mostly ignored by Internet operators and users.

3.1 Benefits of Early Detection

We target our research on the detection of large-scale Internet attacks. In specific, we are interested in early detection of massive worm outbreaks, which could easily lead to a disruption of Internet services. The main
benefits of having an early attack detection system in place are threefold:

1. Network operators and users can be warned early about new attacks, which allows them to take reactive attack defence measures for damage reduction.

2. Hosts and networks that are the originators of a new attack can be identified. In many cases, affected systems can be hardened to proactively prevent future such attacks or outbreaks of the same worm. Attacker identification is also useful for taking legal actions by attack victims against the attack originator(s).

3. Knowing about the worm or attack while it is still active allows to gather comprehensive evidence of an attack (e.g. the worm code or the initial set of infected hosts), which simplifies or in some cases even is a necessity for detailed forensic studies of the incident.

### 3.2 Data Sources for Detection

In principle, any data that contains evidence of an attack can be used for attack detection. Typically, data for analysis is gathered continuously either from hosts or from network elements such as routers, switches and firewalls. Continuously requesting evidence of perceived attacks from end users is in most cases impractical.

- Hosts can provide system and service log files (e.g. Apache web server log, log of failed user authentications reported by sshd etc.).
- Firewalls can provide logs on dropped packets. Some firewalls can also log and block application level attacks.
- Sniffers can capture packet-level traffic (e.g. using tools like tcpdump) of a single host, a network segment or e.g. on links to other networks. For large data volumes and bandwidth, packet-level capturing is technically difficult and expensive.
- Most professional routers and switches can export IP flow traffic reports in the form of Cisco’s NetFlow [36], InMon’s sFlow [107] or hopefully soon in IETF’s IPFIX [73] format. This type of traffic information is easy to obtain as it is commonly used for accounting and network planning purposes by network operators.
3.3 Detection Methods

We now discuss the most commonly used methods for Internet attack detection. These methods are typically used in intrusion detection and prevention systems (IDS/IPS).

3.3.1 Signature Matching

A specific worm typically uses only one or a few exploits to break into other hosts. Once, such an exploit is known, it can be described by a signature. Such a signature is e.g. a short sequence of bytes that matches with parts of received network packets sent by a specific worm or data logged to a system or service log file. For this method, typically packet payload or detailed logs from hosts are required. The term “signature” is used differently in literature. However, it is commonly agreed that a signature exactly describes a feature unique to a specific (usually malicious) event. Some examples that illustrate the large range of signature types are: A network packet from an IP address that is on a blacklist, an illegal TCP flag combination in a TCP/IP packet, an e-mail attachment that has a given hash value, an instant message that contains a given keyword, or more than a thousand DNS requests per minute. There is no clear distinction between a “signature” and an “anomaly”. E.g. the last

- SNMP (simple network management protocol) information from network interfaces is another popular but coarse grained data source. It can e.g. deliver the total amount of bytes sent and received on each network interface of a router, switch or host. Several past security problems of SNMP implementations made network operators hesitant to make this information accessible in certain network environments.

For detecting massive attacks directly in the backbone, packet capturing is too expensive and host information is inaccessible. Therefore, we decided to use IP flow traffic reports for our attack analysis and detection. Chapter 4 will introduce the DDoSVax project, which we co-established to gather and process IP flow (Cisco NetFlow) traffic data from Internet backbone border routers.
example of a DNS request rate signature given could also be regarded as an anomaly.

Current anti-virus software and many IDS/IPS support the use of signatures for detection. Precise signatures offer a low false positive rate but can only recognise previously known attacks, worms and viruses. Most signatures require full network packet information or detailed log files to work.

For our network activity based detection method presented in Chapter 6, a signature of a network activity pattern that can be used to detect the Welchia worm [125] (or one that uses the same propagation method) is described.

3.3.2 Traffic Anomaly Analysis

A traffic “anomaly” occurs if the current network traffic deviates from “normal” (or baseline) traffic. This definition of anomaly is fuzzy in the sense that for general Internet traffic there exists no commonly agreed definition of what “normal” traffic actually is. An alternative notion for the baseline traffic is “attack-free” traffic. However, this does not help any further, as “attack” is also a fuzzily defined term. An anomaly does not have to be security related, it could also indicate the effect of new network services launched or of an event of global interest like NASA’s Pathfinder mission [92] for exploring the Mars that cause a lot of network traffic (also known as “flash crowd”).

In this thesis, we define a traffic anomaly regarding network attacks as traffic that exhibits patterns of a misuse or a disruption of network services. A massive worm outbreak, a few hosts sending millions of spam e-mails, a distributed high-rate ICMP flooding attack that disturbs the routing system, several routers that suddenly crash etc. are examples for such network attack anomalies. Typically, such anomalies are unwanted as they have a negative effect on the reliability and availability of Internet services.

Many past massive Internet worms showed an initial exponential growth behaviour [135] in the number of infected hosts, bandwidth used for their propagation, number of packets sent for scanning the network for vulnerable hosts, number of ICMP unreachable messages generated and other network parameters. Such exponential changes of network parameters can be tracked by monitoring the volume of traffic on a router, the num-
ber and types (scans\textsuperscript{1} vs. sweeps\textsuperscript{2}) of scans occurring, an exponential rise in the number of unique active sources, a significant change in the autocorrelation of events of the same type or in the crosscorrelation of the interaction of two different events. While these are all valid approaches to detect anomalous traffic patterns, no single method can detect all worms or attacks. In addition, automatically calibrating the parameter thresholds based on which to signal a detection for a specific network is still an unsolved research issue. Many parameters show daily rhythms (e.g. bandwidth use by e-mail). In that case, adaptive thresholds that are even more complex to specify correctly must be used.

Our host behaviour based worm detection method presented in Chapter 7 falls into the category of anomaly detection. It uses a combination of network parameters and assigns hosts to behavioural classes depending on their Internet usage. Tracking the number of members in the classes for changes allows to detect network and e-mail based worms.

### 3.3.3 Black Hole or Darknet Monitoring

Another approach for detecting worms and attacks is to monitor activity on a network called “darknet”\textsuperscript{3} that contains routed but unused Internet address space. With “unused” we mean, that no hosts or services are officially announced to exist in that IP address range. If traffic can be seen in such a range, then this can be clearly stated as anomalous as the “normal” baseline traffic is of course “no traffic”. As the traffic entering such a network is typically not responded to, such a network is also called a “black hole” network.

The causes for such traffic can be misconfigurations of hosts or services (e.g. with a wrong IP address for a dependent service), backscatter traffic due to an attack (e.g. ICMP unreachable or TCP SYN replies to spoofed source addresses belonging to a darknet) or scan traffic of e.g. worms that search within all possible IP addresses for vulnerable hosts without a priori knowledge if the target actually exists.

We will give two examples of darknets in the related work section of Chapter 5. Some darknets are implemented not with a passive mon-

\textsuperscript{1} A scan means probing two or more ports on a single host.

\textsuperscript{2} A sweep means probing the same port on many different hosts across a network.

\textsuperscript{3} Our security related use of the term “darknet” has not to be confused with the P2P related use of it that denotes a small friend-to-friend file sharing network.
itoring system but with active honeypots, which somehow contradicts the idea of a darknet as herewith it seems like there would exist valid network services in that darknet.

3.4 Outline

This thesis continues with Chapter 4 (“The DDoSVax Project”), which presents our long-term archive of backbone flow-level traffic data and the infrastructure used to capture and process the Terabytes of data. In Chapter 5, we give detailed forensic analyses of the past massive Internet worms Blaster and Sobig. In Chapter 6 (“UPFrame”) we explain our software framework developed to alleviate the implementation of our near real-time traffic anomaly detection algorithms in the form of plug-ins that concurrently process incoming flow-level data. Finally, we present and validate our host behaviour based worm outbreak detection method in Chapter 7 (“Behavioural Worm Detection”).
Chapter 4

The DDoSVax Project

The DDoSVax [134] project in the Communication Systems Group [1] at ETH Zurich [52] is headed by Prof. Dr. Bernhard Plattner. The project and its two core research members, Arno Wagner and me, are funded by TIK/ETH Zurich, SNF\textsuperscript{1} [117] and SWITCH [4]. Its main focus is security research related to massive attacks that are observable in Internet backbone traffic captured at border routers. While its name “DDoSVax – In search for a vaccine against DDoS attacks” indicates DDoS attacks only, many of its research activities are actually in the field of recent massive worm spreading events, which of course can also have DDoS attack like effects and can be used as preparation for DDoS attacks. Our thesis uses flow-level data captured in the DDoSVax project that were mostly processed on DDoSVax’s cluster computing infrastructure. Therefore, we describe in the following the DDoSVax data archive, the backbone network AS559 that the flow-level data was captured from, the capturing and processing infrastructure and the NetFlow data format used.

\textsuperscript{1}Under grant 200021-102026/1
4.1 DDoSVax NetFlow Archive

As part of the DDoSVax project activities, the complete flow-level traffic of all border routers of the medium-sized Swiss Internet backbone AS559\(^2\) operated by SWITCH is captured since March 2003. The Internet Protocol (IPv4) address range belonging to SWITCH and its customers contains about 2.2 million addresses. AS559 carries around 5% of all Swiss Internet traffic \([91]\). In 2004 on an average working day, we counted roughly 300 GByte traffic in about 60 million flows per hour on the border routers. Network traffic between approximately 200,000 SWITCH-internal hosts and of approximately 800,000 hosts from outside the SWITCH network could be observed per hour. The DDoSVax traffic archive contains roughly 6 TByte of bzip2 \([113]\) compressed unsampled NetFlow data per year and is currently worldwide one of very few with a comparable size and level of detail that are accessible for research. All our worm observations and analyses in this thesis are based on DDoSVax flow-level traffic data unless otherwise stated. The size of AS559 is large enough to get a relevant view of Internet traffic activity and small enough such that captured flow-level traffic traces can still be handled efficiently.

4.2 SWITCH Internet Backbone

The SWITCH Internet backbone connects all Swiss universities (ETH Zurich, EPFL, University of Zurich, University of Geneva, University of St. Gallen HSG, etc.), various research labs (CERN, PSI, IBM research, etc.), federal technical colleges and colleges of higher education (Fachhochschule ZHW, FHA, etc.) to the Internet.

The AS559 network map in Figure 4.1 shows the backbone operated by SWITCH. The core network consists of fibers that connect all major Swiss universities and large cities. Many colleges of higher education and smaller customers are connected to the core network by VPN\(^3\) tunnels through a network operated by Cablecom. Such VPN tunnels are not shown in the map, only the VPN points of presence (PoP) are marked. Four border routers that connect to peer networks exist: One in

\(^2\)AS stands for Autonomous System.
\(^3\)VPN stands for Virtual Private Network.
Zurich, one in Basel and two in Geneva at CERN. Two peerings connect to research/educational networks and another two peerings connect to commercial Internet backbone networks. In the DDoSVax project, only border router traffic is captured at network links indicated by “DDoSVax captured link” icons in the map. One can say that the DDoSVax project only captures traffic crossing the cut of the Internet graph that would completely disconnect AS559 from the worldwide Internet. Traffic from outside the SWITCH network to, e.g., ETH Zurich’s web site can be seen as well as, e.g., e-mail traffic leaving ETH Zurich towards MIT in the USA. However, no traffic between SWITCH customers can be seen – unless if traffic is routed through any of the border routers. Traffic routed through several border routers is reported more than once. According to SWITCH, transit traffic from a computer outside of AS559 to a computer outside of AS559 routed through AS559 is rare. Internal traffic from a source in AS559 to a destination in AS559 routed over the border routers is also rare.
Figure 4.1: The SWITCH academic and research backbone (AS559)
4.3 DDoSVax Infrastructure

Figure 4.2: DDoSVax infrastructure

Figure 4.2 shows the DDoSVax capturing and data processing infrastructure. The four border routers capture the traffic at flow-level. These statistics are exported by the routers in the form of NetFlow records and sent to a NetFlow duplicator. The data is forwarded to SWITCH’s accounting system. In addition, the DDoSVax project gets a copy of the NetFlow records. These records are captured, preprocessed and compressed for long-term archiving. The traffic of the four routers is written to two data files\(^4\) each hour, one with data from the Zurich router and one with data from the three other routers. A few statistics and administrative information are stored in corresponding statistics files\(^5\), which results in a total of four files per hour. The DDoSVax part of the capturing system as well as tools and libraries for offline processing of NetFlow data were designed and implemented by Arno Wagner and described in his PhD thesis [133].

The long-term DDoSVax NetFlow archive resides partially on the hard disks of the nodes and gateway computers of cluster “Scylla”. However, the larger part is stored on magnetic tape in an archive operated by a tape robot. There is also the possibility to forward the current NetFlow data to a computer system running online analysis tools.

\(^4\)Flow data files have names with suffix \texttt{.dat}.

\(^5\)Statistics files have names with suffix \texttt{.stat}.
4.3.1 Cluster Scylla

Offline processing of the DDoSvax archive data is done on the cluster Scylla, which was designed in mid 2003 by Arno Wagner and is located at ETH Zürich at the Department of Information Technology and Electrical Engineering (D-ITET). The cluster is fully operational since January 22nd, 2004. Figure 4.3 shows the front and back view of the cluster. The cluster consists of 22 personal computers assembled from carefully chosen consumer parts. The hardware configuration of each node is:

- Athlon XP “Barton” 2.8 GHz CPU
- 1 GB RAM
- ≥ 120 GB hard disk
- CD-ROM Drive
- 3.5” floppy disk drive
- 1 Gbit/s Ethernet network card

The nodes currently run Debian Linux kernel 2.x with the openMosix [13] cluster software kernel extension, which automatically distributes the processing load by moving running processes to idle cluster nodes from loaded ones. Re-installation of the nodes with Debian Linux images is possible within a few minutes using the fully automatic installation FAI [82] developed at University of Köln, Germany. The nodes are interconnected by a 1 Gbit/s fully switched Ethernet in star topology. Parallelisation for most of our analyses achieved a linear speedup factor of almost the number of cluster nodes used, as we could split the input data by the capture time of the flows and independently calculate the analyses with a small time window overlap and finally merge the results.

In addition to the nodes, there are a few gateway computers and file servers through which one can log into the cluster from the ETH network by using secure shell (ssh). For security and data privacy reasons, this login requires asymmetric cryptographic keys and the cluster nodes have no direct Internet connection.
Figure 4.3: Front and back view of cluster Scylla
Why is the cluster named “Scylla”?  

In Greek mythology, “Scylla” was an attractive nymph and a daughter of Phorcys. One day, the sea-god Glaucis fell passionately in love with her. However, she rejected him. Therefore, Glaucis asked the sorceress Circe to create him a love potion. Bad enough, Circe instantly fell in love with Glaucis, who rejected her. Filled with jealousy against Scylla, Circe poured a potion of herbs into the water, where Scylla was bathing in, and then cast her spell. Suddenly, out of Scylla’s lower body half six monstrous dog heads grew.  

We named the cluster “Scylla” as the cluster’s possibilities are fascinating at first, however controlling its calculation power efficiently is a delicate task. More details (also technical ones) about the cluster can be found on the Scylla web site [6].

4.4 NetFlow  

NetFlow records are created directly on Internet routers by aggregating information from the headers of transported IP packets. Cisco’s NetFlow [36] format version 5 that is used for the DDoSVax traffic archive defines a "flow" as a unidirectional stream of packets between the same two hosts (i.e. IPv4 addresses) using the same protocol (ICMP, UDP, TCP, others) and port numbers. Each flow is reported in a single NetFlow record. The number of packets, the total number of bytes in the network (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. Packet payload is not present in a flow record, which alleviates data privacy concerns. NetFlow records are exported from the routers in the form of UDP packets that contain each a NetFlow header (of 24 bytes) and between 1 and 30 NetFlow records (of 48 bytes each). Flows are exported if a timer expires (i.e. long lived flows are reported every, e.g., 15 minutes), the flow expires (i.e. if inactive for a longer time or end of TCP flow) or if the NetFlow cache is full. Specifics about limitations of and experiences with Cisco NetFlow data can be found in [119, 120]. Tables with the NetFlow v5 header and record format specification can be found in Appendix A.
We give three sample flow records in human readable form:

**TCP**  X.X.X.X:61831 -> A.A.A.A:1025 144 Bytes (3 Pkts)

**ICMP echo request** Y.Y.Y.Y -> B.B.B.B 280 Bytes (10 Pkts)

**UDP**  Z.Z.Z.Z:34371 -> C.C.C.C:53 738 Bytes (11 Pkts)
[15:59:22.703 - 15:59:27.823] = 5.120 s

The IP addresses in above flows were anonymised (as X.X.X.X etc.) and only the most important fields (protocol, source IP and port, destination IP and port, flow size, number of packets, start and end of flow) and the calculated flow duration were shown. Due to the fact that a flow is unidirectional and due to asymmetric routing in the Internet, it is well possible that an outgoing flow sent from ETH Zurich containing a request for a US web site is reported on the Zurich border router, whereas the reply flow containing the web page is reported by one of the Geneva border routers. If bidirectional connections are important for an analysis, the corresponding flows have to be matched first.
Chapter 5

Flow-Level Traffic Analysis of the Blaster and Sobig Worm Outbreaks in an Internet Backbone

5.1 Introduction

In this chapter, we examine worm behaviour from a network centric view based on one of the very rare real backbone traffic measurements of the actual worm spreading events. We analyse two major recent Internet worms: Blaster.A [30], that exploits the Microsoft Windows Remote Procedure Call DCOM vulnerability and which spreads without any user interaction and Sobig.F [31], a worm that installs its own SMTP engine.
and propagates as e-mail attachment, which has to be executed by the user for an infection.

This chapter is organised as follows: We describe our measurement setup and survey related work in the rest of Section 5.1 (Introduction). In Section 5.2 (Network Worm Blaster), the infection steps of Blaster and associated network traffic on packet and flow-level are analysed. In Section 5.3 (E-Mail Worm Sobig.F), we discuss our measurements of Sobig.F related e-mail traffic. Finally, we give our conclusions and an outlook in Section 5.4.

5.1.1 Backbone Measurement Data

For our worm analyses, we used the complete flow-level (Cisco NetFlow v5) traffic of all border routers of the Swiss backbone AS559 operated by SWITCH. Details about this backbone, NetFlow, the DDoSVax project and its flow-level data archive can be found in Chapter 4.

AS559 transit traffic was excluded from our Blaster.A analysis. Traffic routed through several border routers is reported more than once. We eliminated such flow duplicates by counting flows with the same source and destination IP addresses and ports only once within 50 ms. A different method would be to use Bloom filters [21] for this elimination. It is possible that partial loss of NetFlow data during aggregation in the routers and other worm-unrelated larger network events introduced distortions into the plots presented. As we captured all NetFlow traffic exported by the routers and as no other major network events during the analysed time periods were reported publicly, we believe these effects to be small. Another limitation is that no TCP flags are reported in our traces due to constraints in the routers’ hardware-based NetFlow engines.

5.1.2 Related Work

All major anti-virus software vendors published analyses of the Blaster worm code on their web sites (e.g. Symantec [127], Trend Micro [130]) based on a host centric view of the worm’s behaviour. We made use of this information to crosscheck our own measurements with the real worm executable in our testbed.
5.1 Introduction

Symantec has analysed in [126] the infection rate of Blaster in the days after its initial outbreak. According to Symantec, the DeepSight\textsuperscript{TM} IDS sensors reported up to $\sim 1,200$ source IP addresses sending exploit code to 135/TCP per hour during the first week after the outbreak. In total, around 40,000 infected source IP addresses were counted from Aug. 9th to Aug. 16th. It must be considered that only traffic to 135/TCP was used for this analysis.

Long-term archives of network backbone measurement data as we used it for our analyses are rare and difficult to get access to due to privacy laws, data security concerns, the challenge and costs of handling large amounts of real-time statistics data and the possibility of interference with current network operations and accounting.

There are many special-purpose and mostly commercial tools [83] available for processing NetFlow data. Some open source NetFlow tools such as SiLK [34] also exist. Many network operators use such tools to collect NetFlow data for accounting and network planning purposes. They often use only a small sample of all flow records (e.g. 1/400 of all records) and rarely store them for a longer time. We know from several Internet Service Providers and from the network services at ETH that their commercial software used for real-time network monitoring crashed during the Blaster outbreak (mainly due to out of memory errors as a consequence of a huge network activity increase). For long-term capturing and analysis of large amounts of NetFlow data, software is rare. We used the tools developed in our DDoSVax project for capturing and data processing.

The University of California in San Diego (UCSD) operates a “Network Telescope”, which is a /8 subnet that a team of the Cooperative Association for Internet Data Analysis (CAIDA) [28] uses to analyse backscatter traffic of worms and attacks. With this measurement setup one can mostly see traffic due to spoofed source IP addresses and scanning activities. However, traffic of successful infections of productive hosts (especially if a worm uses multiple steps for an infection like Blaster) are not visible in such a passive network setup. CAIDA published analyses of the worms Code-Red, Slammer and Witty [115] but nothing on Blaster or Sobig.F.

Researchers at the University of Michigan and José Nazario from Arbor Networks have observed the Blaster worm using their Internet Motion Sensor IMS [11] in an unused /8 subnet, i.e. in “dark” address space.
They observed a maximum of \( \sim 15,000 \) [12, 94] unique source IP addresses with SYN scans to 135/TCP per hour just after the Blaster outbreak on Aug. 11th, 2003. In our analyses (cf. Figure 5.11), we found around 5,500 such hosts per hour. They used only TCP SYN packets to 135/TCP to derive overall Blaster activity, which is quite imprecise as no distinction between hosts infected by Blaster, by other worms and (possibly uninfected) hosts scanning the network for open ports can be made. In our Blaster analyses in this chapter we show, that the number of exploit code transmissions and of infections were actually orders of magnitude lower than Blaster scanning activity was. When studying their IMS based Blaster activity plot published in [12], we found that between Aug. 14th 0:00 and Aug. 16th 0:00 (GMT) their plot shows almost no Blaster activity. As our measurements could not confirm such a low Blaster activity, we assume that their IMS suffered from data loss during this time (even though this was not stated in their IMS based Blaster publications [11, 12, 94]).

Research on intrusion detection systems (IDS) was done for more than twenty years. However, in an IDS usually a lot about users, resources, running services, and other installed software of the hosts under attack is known unlike to our backbone measurements. Most IDS research focuses on access networks and does not deal with the specifics of flow-level cross-border traffic in backbones.

Several mathematical models [78, 123, 140] were proposed that simulate and predict worm propagation behaviour. From experiences with the worm model and worm infection simulator [135] developed in the DDoS-Vax project, we found this approach to be very limited with respect to characterizing worm outbreaks realistically. This is because it is almost impossible to model effects due to network operators, system administrators and anti-virus vendors intervening during the outbreak and due to a lack of precise data on Internet routing configuration, impacts of congestion, router loads or patch levels of end user and server systems. The parameters of a simulation must be carefully adjusted to each new worm and they are valid mostly only for the very early spreading stage. Due to the scarcity of in-depth analyses of real worm measurements in the backbone, very little about real worm behaviour in the Internet is publicly known.
5.2 Network Worm Blaster

Blaster is a multi stage worm: for a successful infection, six sequential steps, which involve traffic on three specific ports, must be completed. We analysed Blaster for the interplay of infection steps and associated network traffic. We gradually added new restrictions to our traffic filters. This allowed us to differentiate how many infection attempts there were, how many were partially successful up to a specific stage and finally how many were successful. In addition, we analysed our traffic traces for further anomalous behaviour in relation to the Blaster worm.

5.2.1 Outbreak

On August 11th, 2003, the Blaster.A [30] worm was first observed in the Internet. In April 2004, Microsoft estimated the number of all Blaster infected systems since the outbreak to be at least 8 million [84], whereas the Internet Storm Center stated that based on their evaluations of firewall logs provided by thousands of volunteers between 200,000 and 500,000 computers had been infected.

The worm exploited a remote procedure call (RPC) vulnerability of Microsoft Windows 2000 and Windows XP operating systems that was made public in July 2003 by the "Last Stage of Delirium Research Group" in [129] and that is described as critical in the Microsoft Security Bulletin MS03-026 [88]. The same vulnerability (which requires a slightly different exploit code) is present in Windows NT 4.0 and 2003. However, these systems were not targeted by the main Blaster variant Blaster.A.

An infection of a Windows host by Blaster can be prevented by using a firewall that blocks traffic incoming to port 135/TCP and by applying the operating system patch that fixes this RPC vulnerability.

5.2.2 Worm Variants

As no commonly agreed rule exists for worm and virus naming, W32.Blaster.A (Symantec) is also known as Lovesan (F-Secure), W32/Lovesan.worm.a (McAfee), Win32.Poza.A (CA), WORM_MSBLAST.A (Trend), W32/Blaster-A (Sophos), W32/Blaster (Panda) or Worm.Win32.Lovesan (KAV). Besides the A version of Blaster, many more variants were developed based on the same exploit
Blaster infected Host

1. initialise
2. scan
3. transmit RPC DCOM exploit code
4. initiate worm code download
5. download worm code by TFTP
6. execute remote Blaster worm code

transmission (new flow) transmission (aggregated to previous flow)

Victim Host

135 TCP
135 TCP
4444 TCP
launch
launch
Windows
dows
shell
Blaster
worm
Blaster
worm

Figure 5.1: Blaster’s infection steps

code. They differ in the name of the executable or have changed or added mostly malicious functionalities.

5.2.3 Blaster’s Infection Steps

Measurements of Blaster.A infected computer activity in our testbed network was consistent with results of the machine code analysis by eEye described in [49]. The following description holds for Blaster.A, all other variants work very similar. The illustration in Figure 5.1 shows Blaster’s infection steps with a focus on network flows that can be observed. The following subsections explain each infection step in detail and use the same numbering as Figure 5.1 for the steps.

Step 1: Worm Initialisation

When Blaster is launched, it opens a mutex called “BILLY” that is used to prevent multiple infections of the same machine and sets a registry key to assure it is restarted upon each reboot. Then it checks the date. If the current day is the 16th or later or if the current month is from September to December it starts a TCP SYN flooding attack against windowsupdate.com with a spoofed source address, which consists of the two first bytes of the local address and the two last bytes gener-
ated at random. This attack was not successful because Microsoft could simply stop the DNS forwarding from windowsupdate.com to windows-update.microsoft.com. We did not further analyse this attack.

Step 2: Victim Scanning on Port 135/TCP

In Blaster’s initialisation phase, the worm decides whether it will use the exploit code for Windows XP (80% probability) or the one for Windows 2000 (20% probability). According to Symantec [127] the worm then generates an IP address to start scanning as follows: With probability 60%, an IPv4 address of the form $A.B.C.0$ with $A$, $B$ and $C$ chosen at random is used. With probability 40%, an address of the form $X.Y.Z.0$ derived from the infected computer’s local address $X.Y.Z.U$ is chosen. $Z$ is set to $Z$ unless $Z$ is greater than 20, in which case a random value less than 20 is subtracted from $Z$ to get $\tilde{Z}$. Blaster always scans blocks of 20 sequential IP addresses simultaneously. The destination IP address value is incremented by one after each scan.

Step 3: Transmission of RPC Exploit Code

If a TCP connection to destination port 135 can be opened, the exploit code is sent to the victim. If it was vulnerable and the correct exploit code was sent, a Windows command shell process is started that listens on port 4444/TCP and allows remote command execution. Unpatched Windows XP computers automatically reboot within one minute after the RPC exploit code is executed.

According to our measurements with a Blaster.A infected computer in our testbed, the exploit code is sent as a remote procedure call (RPC) “bind” (72 bytes), an RPC “request” (1,460 bytes) and a TCP packet (244 bytes). Summing these values up and adding the size of the headers (40 or 48 bytes for TCP/IP without respectively with TCP options) and also counting the two packets for the TCP handshake, we get 1,976 bytes (without TCP options) or 2,016 bytes (with TCP options) for the whole RPC exploit code transfer.
Step 4: Initiation of Worm Code Download

Blaster then initiates a TCP connection to port 4444/TCP. If successful, the command “tftp -i attacker-IP GET msblast.exe” is executed to start a Trivial File Transfer Protocol (TFTP) download of msblast.exe from the Blaster-infected host. Windows has the TFTP client tftp installed by default.

Step 5: Download of Worm Code by TFTP

If the remote download initiation was successful and the victim’s TFTP requests are not blocked (e.g. by a firewall), the Blaster-infected host is contacted on port 69/UDP for a download of the worm code. The size at the TCP layer of the Blaster.A worm code is 6,176 bytes. In our own measurements with a Blaster.A infected computer, this code was transmitted in 12 TFTP packets of 512 bytes each and a 13th one of 32 bytes. Accounting for each TFTP packet 32 bytes for IP/UDP/TFTP headers, we get 6,592 bytes on the IP layer.

Step 6: Blaster Worm Code Execution

Finally, the Blaster-infected machine stops its TFTP daemon after a transmission or after 20 seconds of TFTP inactivity. In case of success, it sends a command to start msblast.exe on the already open TCP connection to port 4444 of the victim. Now, the victim is running Blaster and starts to infect other machines.

5.2.4 Identification of Blaster Infections by Flows

Infection Stages

We define five different stages A, B, C, D and E that classify to which extent a Blaster infection attempt on a victim host was successful. Table 5.1 lists for each infection attempt stage the required flows (marked by symbol □) and their directions. We use the notation ’A→V’ for a flow from attacker to victim and ’A←V’ for a flow from victim to attacker.

A. The victim host does not exist or does not respond to a connection attempt on 135/TCP.
B. The victim host responds but port 135/TCP is closed.

C. The victim host receives the exploit code but either the exploit code for the wrong operating system was transmitted or the RPC DCOM security patch was already applied.

D. The victim host is vulnerable and the correct exploit code is successfully transmitted to port 135/TCP and the TFTP commands are sent to the remote shell on 4444/TCP but the TFTP server does not respond.

E. The infection is completely successful.

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{Stage} & 135/TCP & 4444/TCP & 69/UDP \\
\hline
A & \checkmark & - & - & - \\
B & \checkmark & - & - & - \\
C & \checkmark & \checkmark & \checkmark & - \\
D & \checkmark & \checkmark & \checkmark & \checkmark \\
E & \checkmark & \checkmark & \checkmark & \checkmark \\
\hline
\end{array}
\]

Table 5.1: Flows required for infection stages A - E

Filtering for Blaster Flows

The infection attempt stages defined in 5.2.4 can be distinguished by filtering our flow-level backbone traffic for the sequential occurrence of specific flows between any two active hosts that contain certain protocols and ports and that have a size and a number of packets in restricted ranges. We derived this information for each of the five infection stages from the Blaster analysis given in 5.2.3, from packet-level measurements in our Blaster testbed, and from tracking flow-level traffic of a host that was infected by Blaster during the actual outbreak and that was highly active on August 12th, 2003.

Obviously, the number of infection attempts per time unit is highest in stage A and lower in stages B to E as the filter criteria get more and more restrictive. This filtering for infection stages shows a reduction in the number of infection attempts of several orders of magnitude as can be seen in the Blaster plots in Figures 5.5 to 5.9.
Challenges of Malicious Flow Extraction

We faced the following challenges when defining our malicious flow extraction filters:

- Retransmissions by TCP due to packet loss (mostly at the receiver) and too short timeouts caused additional packets in the flows; we observed that the initial TCP SYN packets were most likely to be retransmitted (possibly due to an overload of the receiver).
- Different sizes of TCP SYN packets (40 and 48 bytes) due to the presence or absence of the TCP option field of 8 bytes that indicates the maximum segment size accepted.
- Indistinguishability of a TCP SYN packet of a malicious flow and of regular use in case of an unsuccessful connection attempt (e.g. on 135/TCP).
- Inactivity timeouts (30 s) of the NetFlow aggregation engine that cause a split of one flow into multiple flows for slowly responding hosts (e.g. shells on 4444/TCP) requires “glueing” of such flows.
- Preventing to count hosts, which had similar traffic like Blaster but out of order or with non-Blaster payload. Therefore, we also applied a heuristic timing condition, which required that the start time of each flow belonging to an infection attempt must lie within 10 seconds of a first 135/TCP flow seen. In our LAN testbed all such flow start times were below 4 seconds. We chose the larger threshold of 10 seconds due to expected higher delays in a WAN and due to possibly slower processing powers of involved hosts.
- Trivial FTP (69/UDP) used by Blaster is a well-known service. Therefore, we had to further limit our filters to only consider host pairs that had flows on port 135/TCP and 4444/TCP previous to a TFTP transfer attempt.

5.2.5 Blaster Outbreak Traffic Analysis

Our traffic analyses focus on a time span starting on 10th of August 2003, the day before the Blaster outbreak, and ending on 16th of August 2003. In the following plots, we have split the total traffic by origin (from inside
or from outside of AS559). With “inside” we mean all IP addresses of hosts belonging to SWITCH (AS559) and its customers.

For each 5 minute time span, all pairs of hosts that had flows matching the criteria of the infection attempt stages A to E defined in 5.2.4 are identified and accounted to the five stages. Congestion in storage tables of the router’s NetFlow engine can lead to a loss of flows. Therefore, we alleviated the requirements such that only at least one matching flow for each port/protocol type (135/TCP, 4444/TCP, and 69/UDP) needed to be present if the stage required that type at all. However, the effect of this alleviation in the filtering conditions was only minimal.

Infection Attempts

Figures 5.2 to 5.4 show the infection attempts for stages A to C per hour for the time around the Blaster outbreak. Figures 5.5 to 5.9 show the number of infection attempts for each of the five stages A to E defined in Section 5.2.4 per hour for a longer observation time of six days.

Monday, August 11th, 2003 at around 16:35 UTC (see Fig. 5.15) can be regarded as the outbreak of Blaster. We can see in Figure 5.2 that at the outbreak the number of unsuccessful connection attempts to port 135/TCP (stage A) drastically increases from around 0.7 mill. to 1.5 mill. and in the next two hours to 13 mill. flows per hour. In the three hours after the outbreak, the number of stage B infection attempts (victim responding but port 135/TCP is closed) grows from about 50,000 to 1 mill. connection attempts per hour. The number of stage C (Figure 5.4) occurrences jumps from 0 to around 650, while stage D (Figure 5.8) occurrences show only a single host in total during the first three hours of Blaster activity. The very first successful infection from outside to a SWITCH-internal host happened at 17:42 UTC. Quite late, at 18:20 UTC, the first external host is successfully infected from a host in the SWITCH network. In the hour from 17:20 to 18:20, a total of seven infections can be seen in Figure 5.9. More than a full hour passed after the significant increase of port 135/TCP activity and before the first successful infection in the AS559 network happened.

Before August 12th, the vast majority of Blaster traffic originated from outside the SWITCH network. This changed around 6:50 UTC and can be considered as the internal outbreak. Before that, only few hosts within AS559 had been infected. The reason for the delay of the internal...
Figure 5.2: Blaster outbreak: Number of 'stage A' Blaster infection attempts from Aug. 11th 12:00 to Aug. 12th 18:00

Figure 5.3: Blaster outbreak: Number of 'stage B' Blaster infection attempts from Aug. 11th 12:00 to Aug. 12th 18:00

Figure 5.4: Blaster outbreak: Number of 'stage C' Blaster infection attempts from Aug. 11th 12:00 to Aug. 12th 18:00
5.2 Network Worm Blaster

Figure 5.5: Number of 'stage A' Blaster infection attempts from Aug. 10th to Aug. 15th

Figure 5.6: Number of 'stage B' Blaster infection attempts from Aug. 10th to Aug. 15th

Figure 5.7: Number of 'stage C' Blaster infection attempts from Aug. 10th to Aug. 15th
Figure 5.8: Number of 'stage D' Blaster infection attempts from Aug. 10th to Aug. 15th.

Figure 5.9: Number of 'stage E' Blaster infections from Aug. 10th to Aug. 15th.
outbreak is that the external outbreak happened not during Swiss work
time and most internal Windows hosts were switched off during the night.
In the time 23:55 on Aug. 12th to 2:01 UTC on Aug. 13th, only a single
internal host was “successful” and infected 11 outside hosts that were all
in the same /16 network range.

In Figure 5.15 and in the plots for stages A and B, we can observe a
drop in the number of connections from external hosts from 08:30 to
09:10 on August 12th. This was caused by an inbound port 135/TCP
filter installed at one border router of AS559. As soon as the filter is
released, the number of Blaster related connections per 5 minutes jumps
to an even higher value than before filtering. Filtering only incoming
traffic on one of the routers merely blocks attackers from outside trying
to infect further internal hosts using this incoming link. Infections over
other incoming links as well as the spreading of the worm outside of
AS559 continues unaffected of this filter. We can observe another but
smaller drop of infection attempts in Figure 5.14 coming from external
hosts, decreasing since 2:40 on August 13th with its lowest point around
5:00. This is most probably also an effect of manual port filtering.
The first peak of stage C is between 9:20 and 10:20 on August 12th,
with around 15,000 infection attempts. Our analysis showed that around
70% of the stage C infection attempts in that time span came from one
single /16 network. The vast majority of the victims of these infection
attempts were in the next higher /16 network lying outside of AS559.
These connections were probably generated by Blaster scanning the local
subnet, but the scanned addresses were constantly increased by one and
suddenly pointed out of the local subnetwork and therefore the infection
attempts were routed over the SWITCH border gateways. At the same
time span the infected hosts of that subnetwork generated only 29 stage
D and not a single stage E infection. The reason for this lack of successful
infections may be that in the destination subnet the hosts were already
patched.

Many similar IP-neighbourhood attacks happen during the second sig-
nificant increase of stage C occurrences starting around 15:20 on August
14th. The majority of attacks originate in one single /16 network and
most destinations are in the next higher /16 network lying outside of
AS559. In that network, most hosts were apparently also already patched
as almost no successful infections were observed. We can deduce, that
choosing as backscatter or honeypot network one with IP addresses ad-
The reason why these scans show up as peaks in the plot is probably that most of the hosts were infected in a small time range internally and therefore started their scanning around the same time. Consequently, they also reach the next network at about the same time and when they have passed the address space of that subnet, they came probably to a network less populated or with some filtering, which caused a drop of stage C infection attempts. Their scanning then appears as stage A or stage B.

The plot of stage C in Figure 5.4 shows a small peak of 631 infection attempts on August 10th in the hour of 19:20 - 20:20 UTC before the massive outbreak of Blaster. A single host becoming active around 19:40 is responsible for 80% (522) of these attempts. At that time, the exploit code used by Blaster was already published. From that specific IP address we observed a scanning of port 135/TCP and for the addresses that the scanning was successful the exploit code was sent. It is possible that this was some testing in the development phase of Blaster, but more likely someone just tried out the exploit code for fun or for some abuse.

### Successful Infections

The stage E plot of successful Blaster infections shows a peak at the right end with 35 infections within 3 hours, from 21:20 on August 15th to 0:20. 29 of these infections originate from one host and have their victims in the same /17 network range. This host obviously scanned by chance a network with many vulnerable hosts. A surprise in Figure 5.9 is, that despite the high number of Blaster infected hosts worldwide, we can only observe very few successful infections going over SWITCH’s border routers. Over the analysed time period, from the outbreak, on August 11th, to August 16th, 0:20, we observed only 215 successful infections in total. 76% of the observed infections originate from within AS599 and 24% are from external hosts. 73 different hosts have successfully infected others. The reason for this low number is that the vast majority of successful infections happened within the local networks and did not cross the backbone border routers. The ten most successful hosts have performed 138 (64%) of all observed infections. The hosts in the top ten list scanned networks with high numbers of vulnerable computers. The
47 infections of the "winner" all go to addresses in a range of 13 adjacent /16 networks. The fact that 11 out of the top 21 infecting hosts were found to be in the same /16 network is an evidence that this network suffers from slow patching procedures.

5.2.6 Worm Code of Multi Stage Worms: Low Frequency vs. High Threat Potential

From our border router backbone measurements we conclude that for multi stage worms, which use several different steps before actual worm code is transmitted, the number of observable hosts that successfully infect others is extremely low (4 hosts during the initial outbreak per hour in Fig. 5.9). This is in heavy contrast to the high number of hosts scanning for vulnerable hosts in Fig. 5.5.

The design of Blaster relies on three different connections, one of which was on a port rarely used (4444/TCP) and the other involved a modestly popular service (TFTP). As such connections are filtered in many networks, this is a source of errors. It must also be considered that 40% of the scans generated by Blaster targeted addresses in the same /16 network as the infected computer as discussed in 5.2.3. As spreading within the same network is much faster than over the backbone, especially regarding delay, most computers were successfully infected from others located in the same or nearby networks. Such intra network traffic as well as traffic between AS559 customers are not observable in the DDoSVax NetFlow cross border traffic data that we used.

As a consequence, actual worm code (but not exploit code) transmissions are quite rare when observed at the backbone border. We are not aware of any other publications containing measurements of actual successful Blaster worm code transmissions (and not simply observing scanning traffic to 135/TCP) over the border of an AS. This low worm code transmission frequency observed has consequences for, e.g., sampling for malicious code in a backbone, as sampled packet sniffing will almost never capture real worm code. Automatic detection of such worm code would consequently require huge storage and computing resources as only full unsampled traffic capturing can catch the worm code. Furthermore, automatic blocking of source hosts infected by a multi stage worm becomes almost infeasible as the sample of suspicious traffic of some of the later stages probably will be too small for reliable auto-
matic filter creation. Missing even a single successful host infection in the backbone will destroy the effectiveness of backbone border router worm filtering efforts. Even a very low frequency of malicious worm code occurrences in the backbone border traffic has apparently still a high threat potential. However, even for multi stage worms, observation of cross-border backbone traffic can yield valuable information for early worm detection as at least the scanning and exploit code activities are clearly visible.

5.2.7 Coarse Grained Analysis

Due to the huge number of possible combinations of protocols, ports, sizes and number of packets involved in a new worm outbreak, it would be very resource consuming to constantly watch host activity for new worms on such a fine grained level as we used it in our Blaster infection attempt analyses. Therefore, we also present the Blaster outbreak on a more coarse grained level disregarding size and numbers of packets per flow constraints in the remainder of this section.

**ICMP Measurements**

![ICMP packets per hour](image)

Figure 5.10: **Blaster worm: ICMP packets per hour**

The graph in Fig. 5.10 shows the total number of ICMP packets per hour. The ICMP traffic sent from hosts within AS559 and from outside of AS559 are shown separately. We noticed approximately a fivefold increase in the rate of ICMP packets per hour sent from AS559 and a twofold increase for the rate of ICMP packets per hour sent in total during peak time compared to the base level before the outbreak. This
large backscatter traffic of ICMP messages can be explained by many
destination unreachable notifications caused by unsuccessful connection
attempts to blocked ports and non-existent hosts.

Activity on Port 135/TCP

The graphs in Fig. 5.11 and 5.12 show the number of unique IPv4
source addresses from which connections to port 135/TCP were initi-
ated. Source hosts are separated into 1) all hosts, 2) hosts within AS559
and 3) hosts outside AS559. The plots use aggregation over one hour
respectively 5 minutes observation intervals to build the set of active
hosts. The brackets [ and ] in the hour plots indicate the smaller time
window used in the plots with 5 minutes resolution.

\[0.0 \times 10^0\]
\[1.0 \times 10^3\]
\[2.0 \times 10^3\]
\[3.0 \times 10^3\]
\[4.0 \times 10^3\]
\[5.0 \times 10^3\]
\[6.0 \times 10^3\]
\[1.0 \times 10^3\]
\[1.2 \times 10^3\]
\[1.4 \times 10^3\]
\[1.6 \times 10^3\]

\[10.08.\]
\[11.08.\]
\[12.08.\]
\[13.08.\]
\[14.08.\]
\[15.08.\]
\[16.08.\]

\[0.0 \times 10^0\]
\[2.0 \times 10^2\]
\[4.0 \times 10^2\]
\[6.0 \times 10^2\]
\[8.0 \times 10^2\]
\[1.0 \times 10^3\]
\[1.2 \times 10^3\]
\[1.4 \times 10^3\]
\[1.6 \times 10^3\]

\[10.08.\]
\[11.08.\]
\[12.08.\]
\[13.08.\]
\[14.08.\]
\[15.08.\]
\[16.08.\]

\[0.0 \times 10^0\]
\[2.0 \times 10^2\]
\[4.0 \times 10^2\]
\[6.0 \times 10^2\]
\[8.0 \times 10^2\]
\[1.0 \times 10^3\]
\[1.2 \times 10^3\]
\[1.4 \times 10^3\]
\[1.6 \times 10^3\]

\[10.08.\]
\[11.08.\]
\[12.08.\]
\[13.08.\]
\[14.08.\]
\[15.08.\]
\[16.08.\]

**Figure 5.11:** Blaster worm: Unique source addresses per hour

**Figure 5.12:** Blaster worm: Unique source addresses per 5 minutes

We observed around 140 hosts/hour connecting to port 135/TCP in the
hours before the outbreak. There is an interesting small peak of 327
hosts/hour on Sunday, August 10th, 2003, 18:00-19:00 UTC indicated
with an arrow. Figure 5.12 shows that this peak stems from a single five
minute interval starting at 18:35. In this interval, 263 hosts connect to port 135/TCP. We assume that the peak was either a preliminary version of Blaster that was used to test-run a generation limited infection or that it was a scan to identify an initial host population to be infected or someone just playing with the RPC DCOM exploit code. There might have been more such tests, but they are less visible. From the stage C
analysis in Section 5.2.5, we remember the increased infection attempt activity also involving 4444/TCP connections around 19:40 - 20:10 UTC. The primary Blaster outbreak, which is indicated by a small vertical arrow on the time axis in all Blaster plots, starts on Monday, August 11th, 2003, around 16:35 UTC with 64 hosts from outside AS559 connecting per 5 minutes (Fig. 5.12), but increases to 96 hosts at 16:55 and then sharply to 832 hosts active per 5 min at 18:15. A rather chaotic phase without major increase follows. The hour plot shows a peak of about 5,500 hosts scanning on 135/TCP on 11th during the hour 19:00-20:00. The number of active source hosts increases again when hosts within AS559 begin to contribute in the time span August 12th, 2003, 6:00-7:00 (Fig. 5.11), reaching 1,030 active internal hosts per hour in the time span 11:00-12:00. Figure 5.12 shows that around 6:50 (8:50 CEST) many hosts within AS559 became infected by Blaster. This can be explained by the start of the workday in Switzerland. We assume that most of the vulnerable hosts in AS559 were not running at night, which explains the late internal outbreak as well as the clearly visible circadian activity pattern.

Another remarkable event also present in the plots for the infection attempts of stages A and B is the sudden drop of outside connection attempts on August 12th, 2003, 8:30-9:10, which can be seen in Figures 5.13 and 5.14 that show the rate of flows to 135/TCP. This drop is due to temporary blocking of port 135/TCP on some of the AS559 border routers for incoming connections by SWITCH. This ingress filter proves mostly ineffective as a countermeasure to stop the fast increase in the number of new internal host infections. However, if a complementary egress filter were installed at the same time as the ingress filter was activated, this would have prevented up to a thousand AS559 internal hosts per hour from trying to infect AS559 external hosts. A similar filtering effect can be seen around 2:40 on the 13th. This port filter is also only partially effective.

Activity on Port 4444/TCP

As explained in Section 5.2.3, a successful transmission of the exploit code to 135/TCP makes Blaster connect to 4444/TCP, where it tries to initiate a remote download. Figure 5.16 shows the number of flows per hour to destination port 4444/TCP. Several significant short peaks of
scan traffic to this port from hosts outside of AS559 can be seen. An analysis with 5 minute buckets revealed that these traffic peaks were constrained to 15-20 minutes of high activity and that the number of unique source IP addresses connecting to port 4444/TCP did not show significant changes during these flow peaks. We conclude that the first flow peak might result from a pre-test of a propagation-limited Blaster-like worm, and the other peaks might result from a few network operators scanning the network for open 4444/TCP ports.

Figure 5.16: Blaster worm: Flows to 4444/TCP per hour

Activity on Port 69/UDP

A Blaster victim that has received the correct RPC exploit code and the TFTP download command initiates a connection to the Blaster-infected host on port 69/UDP and tries to download the worm code with the trivial file transfer protocol (TFTP). Hence, we expect to see many connections with little payload to this port containing mainly the TFTP commands to fetch the worm code. If this is successful, we should see larger amounts of data being sent from source port 69/UDP of the Blaster-infected host back to the victim.

The plots of bytes per hour to destination port 69/UDP in Figure 5.17 shows a base level of about $15 \cdot 10^3$ to $20 \cdot 10^3$ bytes per hour. There is a huge peak of $2.5 \cdot 10^5$ bytes in the hour from 16:00-17:00 on August 10th. For 92% of this peak traffic, hosts from AS559 are responsible. The plot for the traffic originating from port 69/UDP in Figure 5.18 reveals that these connections were apparently unsuccessful as almost no data was downloaded. It also shows (indicated by the first arrow from left) that between 18:00-19:00 on 11th worm code was almost exclusively
downloaded from hosts outside AS559. With a two hour delay (indicated by the second arrow from left), worm code is almost exclusively uploaded. However, these peaks of roughly 70,000 bytes each only account for about 10 worm code copies of 6,592 bytes transmitted during each peak. The third arrow from left in the plot of Figure 5.18 indicates, that after Aug. 12th 23:00 the vast majority of total bytes transmitted from source port 69/UDP was sent from infected hosts within AS559 to outside hosts. The analysis of the activity of unique source addresses sending traffic to destination port 69/UDP as shown in Figure 5.19 reveals a peak of about 160 unique IP addresses that were involved in the probable pre-test phase of the worm. About 250 flows with an average size of 1.4 kB go to port 69/UDP from AS559. Figure 5.17 shows the increased bytes per hour activity. The small number of 1.5 flows per involved host on average indicates that this was not UDP scan traffic to port 69/UDP as one might have expected but rather small file transfers.
Figure 5.17: Blaster worm: Bytes to 69/UDP per hour

Figure 5.18: Blaster worm: Bytes from 69/UDP per hour

Figure 5.19: Blaster worm: Unique source addresses (dest. 69/UDP) per hour
5.3 E-Mail Worm Sobig.F

5.3.1 Outbreak of Sobig.F

On August 19th, 2003, the Sobig.F [31] e-mail worm that runs on Microsoft Windows 95 or higher first appeared in the Internet. Besides spreading via executable e-mail attachments of varying sizes and providing its own message transfer agent (MTA) for sending e-mail, the worm is programmed to update itself at predefined dates by downloading new code from predefined computers. By timely intervention of network operators and system administrators this update mechanism could be blocked by shutting down all 20 predefined servers. The original worm was programmed to disable itself on 10th of September 2003. Date and time are taken from a small set of hard-coded IP addresses of global time servers (NTP). The e-mails sent use an arbitrary sender and recipient address taken from local files of the infected host. This type of social engineering is obviously intended to fool users to open attachments seemingly from people they know.

The graph in Fig. 5.20 shows the total number of bytes per hour transmitted as e-mail (SMTP) traffic over the SWITCH border routers. A daily rhythm can clearly be seen. The five workdays have rather heavy traffic with a maximum around 5 GByte per hour, whereas on Saturdays and Sundays the traffic is considerably less. The lunch break can be identified easily during weekdays.

On Tuesday, August 19th, 2003 there is a huge increase in bytes transmitted over SMTP that rises up to around 21.7 GByte/hour at 12:00-13:00 UTC, which is four to five times more than ordinary. This can be regarded as the peak of the Sobig.F worm spreading event. In Section 5.3.2 we will see, that Sobig.F’s outbreak was around 9:00 UTC. The SMTP bandwidth usage plot clearly shows that the vast majority of the border e-mail traffic during the massive worm spreading is originating from within AS559.

The graph in Fig. 5.21 shows the number of flows per hour split by origin of the e-mail sender. Interestingly, late on Monday 18th of August 2003 there is a short peak of flows from outside AS559. We found that the number of unique hosts did not rise significantly during this peak. Therefore we assume this to be scanning traffic for SMTP hosts originating from a few hosts only. The reason for this peak could be a massive scan
for initial SMTP target servers for Sobig or an activity of a Spammer. During the actual outbreak, the number of unique hosts sending e-mail from AS559 shows significant peaks.

![Figure 5.20: Sobig.F worm: SMTP traffic volume per hour](image)

**Figure 5.20:** Sobig.F worm: SMTP traffic volume per hour

![Figure 5.21: Sobig.F worm: SMTP flows per hour](image)

**Figure 5.21:** Sobig.F worm: SMTP flows per hour

### 5.3.2 Identification of Sobig.F E-Mails

In our NetFlow flow-level data, normally one flow corresponds to one e-mail delivered by SMTP. This is because subsequent IP packets containing SMTP data sent from a message transfer agent (MTA) to an SMTP server are typically aggregated by a router to a single flow. We used the size of Sobig.F infected e-mails to filter out Sobig.F e-mails from the total SMTP traffic observed.

#### The Testbed

In order to observe Sobig.F traffic at packet-level we used a testbed with an attacking host (Sobig.F on Windows XP) and a server (see Fig. 5.22). On the server (Linux Fedora Core1) we installed the services NTP, MTA
and DNS. Sobig.F uses a hard-coded list of NTP servers to check that the current date is earlier than September 10th, 2003 before activation. We chose 129.132.2.21 for our server from this list. The DNS service (bind) was configured to resolve all name queries (for A and MX DNS records) from the attacker to the server IP address (129.132.2.21) so that the e-mails from Sobig were all sent to the MTA (sendmail) running on our server. The packet capturing was done on the server machine.

**Observed Worm Transmissions**

In the testbed we captured the packets of several successful Sobig.F transmissions and observed an average of about 100 port 25/TCP packets sent from attacker to MTA with a total size (including IP and TCP headers) of about 104,000 bytes. The flows in the other direction consisted of about 40 packets with a total size of about 2,000 bytes.

For e-mails rejected by black- or whitelist spam filters, the flow from attacker to MTA consists of 8 packets with a total size of about 400 bytes, while the flow in the opposite direction shows 11 packets with a total size of about 850 bytes.

**Flow Size Distribution**

The large worm size of about 100 kB and the aggressive spreading algorithm caused many retransmissions by TCP during Sobig.F propagation as can be clearly seen in Figure 5.23 that shows the two histograms of e-mail sizes for a one hour time span during Sobig.F (on August 19th, from 12:20 to 13:20) and for the same hour the day before. Please note that the values of the graph for the day before were multiplied by five to

![Figure 5.22: Testbed for Sobig.F](image-url)
Figure 5.23: SMTP flow size distribution before and during Sobig.F

Figure 5.24: Number of Sobig.F transmissions over time

make them better visible. The wide peak of successfully transmitted Sobig.F e-mails, starting at about 103,000 bytes then decreasing to about 109,000 bytes but still being significant up to about 125,000 bytes can easily be seen.

Further analyses showed that there were about twice as many flows of size 0 - 1,000 bytes (probably rejected e-mails) during the initial outbreak hour as compared to the day before. There were also some noticeable sharp peaks between 4,800 bytes and 5,100 bytes. Further analyses showed that all these peaks originate from flows with only two source addresses in the same subnet. As Sobig.F infected hosts could be used as mail relays, these servers might have been abused for sending spam.

Number of Worm Transmissions

Figure 5.24 shows the plot of the number of flows with a size between 103,000 and 125,000 bytes, from which we can assume that they originate from successfully transmitted Sobig.F e-mails. As the number of e-mails in that size range was 350 on August 18th, 12:20 to 13:20 and starts to rapidly increase on August 19th at about 9:00, this can be regarded
as the Sobig.F outbreak. The number of successful transmissions raises drastically until about 12:00 and then starts to decrease until the end of the workday at about 18:00. The peak of 137,000 transmissions on August 19th is by far the highest, on August 20th the peak reaches about 100,000 and on August 21th up to 50,000 transmissions per hour were counted. The decreasing heights of the peaks can be explained by people updating their anti-virus software and cleaning up their machines.

5.4 Conclusions and Outlook

Our observations have shown that spreading events of massive worms can clearly be seen in traffic statistics derived from flow-level traffic information of Internet backbones. Worms are a major threat to the reliability and availability of Internet services, especially as those worms seen so far did not aim at disrupting Internet services by launching attack code but merely aimed at fast and worldwide propagation.

We have seen some indication for test runs or preliminary scanning several hours before the actual Blaster outbreak. One consistent effect in all our observations is the time-skew between incoming infection traffic and infection traffic originating from AS559. This was mostly due to the fact that most vulnerable Windows computers were switched off during the night and consequently could not be infected and that e-mail worms like Sobig.F require the attention of a user (e.g. for executing an attachment) for an infection. In addition, this also depends on the scanning and spreading strategy. The observed time window could have been used for taking preventative countermeasures if an early detection system for new worms had been available in backbone networks.

Blaster is a multi stage worm, which uses several protocols and connections for data exchanges before actual worm code is transmitted. Our analyses have shown that this multi stage nature together with Blaster’s rather strong probability of 40% for vulnerable host scanning in the same /16 network as the infected computer is located has surprising consequences: only very few successful host infections and consequently almost no worm code can be detected (and possibly filtered) in the backbone traffic. Nevertheless, these few successful infections over the international backbone had devastating consequences for many Internet users and stub networks. Consequently, automated effective blocking of actual
worm code on backbone border routers is almost infeasible as only a few missed worm code transmissions will completely destroy the success of such security efforts. Furthermore, automated efficient capturing of new worm code in the backbone becomes a challenge due to the scarcity of such transmissions (this holds at least in the case of Blaster). The few worm code instances observed are not to be confused with the heavy scanning traffic and the DCOM RPC exploit code that is sent in the first steps of a Blaster infection and which were transmitted quite frequently. In addition, the ineffectiveness of ingress port blocking filters on routers in the hope to stop a further increase of internal infections was illustrated for Blaster. It was also shown, that AS559-external networks with IP addresses adjacent to AS559-internal networks were more heavily attacked than others due to Blaster’s incremental scanning algorithm. Choosing as backscatter or honeypot network one with IP addresses adjacent to small internal subnetworks can help reducing the time for detection of worms that scan IP addresses linearly increasing.

Several challenges to extracting actual malicious traffic at flow-level were stated such as the sporadic use of a TCP option field for the maximum segment size in SYN packets that enlarges the packet header and frequent packet retransmissions by TCP that both let the measured flow vary in size and consequently lower the accuracy of simple flow size filters. Finally, we discovered that 11 hosts of the top 21 successfully infecting hosts were in the same /16 network, which is an evidence that this specific network suffers from slow patching procedures.

From our findings, we can draw some lessons for network operators:

- Mere ingress filtering on one of the AS border routers cannot stop a worm from spreading. Filters that should stop worms (e.g. block traffic with a specific protocol and destination port like 135/TCP or 4444/TCP for Blaster) should always be applied on ingress and on egress of a router. This also prevents infection traffic to leave the operated network, which helps to reduce global worm activity. Installing them also on routers within an AS prevents the worm from spreading between networks operated by customers of a backbone provider and hence increases the filter effect even more. Of course, such filters will break all network services that rely on traffic that is filtered.

- Even if a worm is spreading massively in the worldwide Internet,
this does not automatically mean that many AS internal hosts were already infected as we have seen when pointing out the time-skew between the external and the internal Blaster outbreaks. Consequently, faster global communication of new outbreaks between network operators would help to make use of this time-skew for deploying countermeasures in order to reduce damage.

- New massive worms do not stringently appear without any prior signs of warning. Continuously monitoring the network for suspicious activity could have revealed the limited spreading Blaster-variant we detected or a significant change in the e-mail flow size distribution during Sobig and proactive countermeasures could possibly have been deployed.

- Networks with slowly patched hosts can become visible as active attack sources during a worm outbreak. Better maintenance of such hosts and networks should be enforced or alternatively, such networks might be temporarily partially disconnected from the rest of the backbone by a backbone operator if the attack does not cease within reasonable time.

- The large number of hosts infected by Blaster and other worms reveals that many client computers are poorly protected. They offer their network services to neighbouring computers or even worldwide with its user neither being aware of this nor using this functionality. A more secure default configuration of client operating systems (i.e. a very restrictive firewall activated by default) as partially introduced in August 2004 with Microsoft WinXP’s Service Pack 2 that comes with a software firewall would help to reduce future infections by network worms.

- A more restrictive configuration on Internet uplinks to only offer predefined services and others (e.g. TFTP) only upon customer request would have prevented many Blaster infections as for a successful propagation also less popular protocol/port combinations like 4444/TCP were required.

As a consequence of the Blaster and Sobig.F analyses, we developed algorithms for early detection of worm outbreaks in the backbone that were successfully validated on past worms captured in the DDoSVax
Several detection methods are presented in Chapter 6. One very promising method based on a classification of the hosts’ traffic behaviour [45] is discussed in Chapter 7.

Further research on and measurement analyses of worms and countermeasures are vital for a better understanding of worm propagation and the development of effective countermeasures, especially as worm authors get more and more sophisticated.
Chapter 6

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

6.1 Introduction

The number of security incidents each year reported by CERT/CC grew exponentially from 6 in 1988 to 137,529 in 2003 [33]. Recent massive Internet worm outbreaks such as Blaster [127], Nachi [86], Witty [132] and Sasser [128] have shown that millions [84] of hosts 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.
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. We are convinced that monitoring traffic activity in the backbone for massive attacks like worm outbreaks or DDoS attacks and hence being able to issue early warnings to corporate Internet users and to react in a timely manner increases the availability of Internet services and reduces economic damage.

In this chapter, we present our open source near real-time backbone monitoring framework named UPFrame (pronounced “up-frame”). Section 6.2 (UPFrame Framework) explains its architecture, buffer management, plug-in support, traffic shaping algorithm, and applicability. Section 6.3 (Worm Detection Plug-ins) discusses several plug-ins that we developed for online monitoring of high-speed backbone traffic in order to detect worm outbreaks. We demonstrate the effectiveness of our algorithms by validating the implemented plug-ins on backbone traffic captured in the DDoSVax project with the outbreaks of the Internet worms Nachi and Witty in Section 6.4 (Analysis of Real Worm Traffic). The chapter finishes with a discussion of the results in Section 6.5 (Discussion) and our conclusions in Section 6.6 (Conclusions and Outlook).

6.2 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 $\sim$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 open source software and can be downloaded from the UPFrame web site [111].

Any UDP packet stream sent to a fixed port can be captured and processed by UPFrame. Our framework provides a parsing library for NetFlow v5 UDP packets. However, UPFrame is not restricted to process NetFlow data packets only.

6.2.1 Architecture

Figure 6.1 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.
6.2.2 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 6.2. 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 time-out 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 6.2.3), which explains the different read positions of the plug-ins in Figure 6.2.

[Image: UPFrame buffer handling diagram]

Figure 6.2: UPFrame buffer handling
6.2.3 Traffic Shaping

The traffic shaper is realized as a low pass filter on the incoming traffic rate and uses the leaky-bucket [55] 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

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

with \( t_{\text{out}} \) 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. \( t_{\text{out}} \) 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 \( t_{\text{max}} \) and the just calculated value limits the maximum speed for buffer 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 [110].

Figure 6.3 shows bursty incoming NetFlow UDP data traffic exported by one SWITCH router on a day with high Sobgig.F worm activity.
Between the traffic bursts there are two to three seconds of inactivity due to the fact that the router buffers outgoing NetFlow data. We used UPFrame's traffic shaper on a UDP forwarding plug-in. The resulting data rate at a receiver of the shaped and forwarded UDP NetFlow data traffic is also shown in Figure 6.3. The traffic bursts were eliminated and the shaped output oscillates in a small range around $340 \cdot 10^3$ Bytes/s instead of between 0 and $590 \cdot 10^3$ Bytes/s. Network socket buffering by Linux at the receiver also contributed to the oscillation effects seen.

![UPFrame traffic shaping on bursty router NetFlow data exports](image)

**Figure 6.3:** UPFrame traffic shaping on bursty router NetFlow data exports

### 6.2.4 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.

6.2.5 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 also 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 low memory and processing footprint.

6.2.6 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. From the DDoSVax project, there exist 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 a UDP packet processing framework as its name indicates. It is extensible and light-weight and it is 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 6.3, we give some sample use cases of the framework for worm outbreak detection. A test installation of the framework at SWITCH (AS559) for online network
monitoring during several weeks confirmed the framework’s stability. In early 2005, P2P traffic was monitored online using current incoming DDoSVax NetFlow data for a few months for developing and validating a P2P heavy hitter population tracking algorithm. The validation was time critical as P2P application layer polling was used for confirming the P2P network of a new P2P node just found (i.e. sending a handshake-message of the presumed P2P protocol and check the reply to see if a host actually speaks that P2P protocol on the identified port).

6.2.7 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 [27] 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 [104], which was implemented with Perl scripts and modules and was optimised to provide near real-time traffic bandwidth usage plots with RRDTool [99] split by protocol type and port. No development seems to have happened after 2003. Mark Fullmer’s flow-tools [59], last updated in March 2003, are a collection of NetFlow tools for capturing, storing, filtering and reporting. Peter Haag from SWITCH [4] released nfdump [64] and NfSen [65] in 2004 under the BSD licence. These tools for capturing and processing NetFlow data are particularly useful to profile specific hosts, create top N statistics and extensively filter NetFlow data. Researchers from Intel and collaborators from ten universities are developing COMO [72], an open infrastructure for network monitoring that will allow customized traffic queries that run continuously on live captured data streams, and retrospective queries that analyze past traffic data to enable network forensics. The fourth alpha version of COMO was released on June 19th, 2005. COMO is targeted to be a multipurpose network monitoring framework, which can work together with multidimensional indexes for network diagnosis (MIND). This allows to build and query indexes of large amounts of captured data distributed over several monitoring nodes. The fourth alpha release of COMO does not provide a traffic shaper mechanism for its plug-ins as UPFrame does.
Some packet-based network monitoring systems such as ntop [40] also provide NetFlow support. The nProbe extension of ntop can act as NetFlow aggregator that emits Cisco-like NetFlow records from packet captures and calculates some basic statistics on the received flows. Those packet-based tools were developed and optimised for monitoring local area networks and not backbones.

6.3 Worm Detection Plug-ins

High-speed Internet links have become a commodity. Huge amounts of data are transferred daily but only little is in fact 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 AS559 border routers in near real-time and store a log of the calculated traffic statistics. Interactive scripts running on a standard web server like Apache [9] 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 chapter, we will apply them to monitor the outbreak of large-scale Internet worms. The host behaviour based detection method, which was very successful in detecting e-mail and network based worms, is discussed separately in Chapter 7.

6.3.1 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 Figures 6.5 (incoming TCP traffic during 24 hours with normal network operation) and 6.7 (incoming ICMP traffic before and during the Nachi worm) are organised in stripes of neighbouring IP addresses. Each pixel represents the activity of a specific IP address (or alternatively of an IP range with a given network prefix length). The activity map has several columns. In each row of a column, the observed IP addresses are incrementally ordered. The activity of the lowest IP address is represented by the pixel on the left side, the activity of the highest one on the right side of the row. The activity pixel of the next higher IP address is on the left side of the next row as can be seen in Figure 6.4.

If hosts with neighbouring addresses are very active, they can be seen as a line of lighter pixels (see Fig. 6.5 for such active networks). Black pixels indicate no traffic at all (see Fig. 6.5). This can be due to the inactivity or inexistence of hosts for that IP address range or e.g. due to some IP address ranges not being routed by intent. The number of bytes for each flow is accounted once to the sending host as outbound and once to the receiving host as inbound. The five most active hosts are highlighted with white circles.

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 $2^{32} \cdot (4 + 4) = 32$ GB. As this would be problematic in main memory, we use a hash table and store only the traffic of hosts observed with activity in the currently processed intervals.
6.3.2 Port Usage Statistics

We also filtered and counted the number of flows for a given TCP or UDP source or destination port, and counted the number of unique IP addresses using a service specified by a port and protocol as shown in Figure 6.6. Such port usage statistics 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 restrictive port filters are less suited for the detection of unknown worms.

6.3.3 Further Plug-ins and Analyses

Arno Wagner developed a plug-in that uses a generic method [136] for worm outbreak detection based on the compressibility of certain network traffic parameters. In Section 6.4.2, we illustrate how it detects 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 developed in DDoSVax [134] student theses. 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, the network services group at ETH Zurich uses UPFrame 24/7 with a plug-in for monitoring P2P heavy hitters [66].
6.4 Analysis of Real Worm Traffic

6.4.1 Nachi Worm

The Nachi worm [86], also known as Welchia worm [125], 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 6.6: Blaster and Nachi worm outbreaks](image)

**Activity Based Analysis**

Amazingly, the number of unique source IP addresses, which traffic to port 135/TCP per hour originated from, was more than three times higher while Nachi was active compared to during the Blaster outbreak as can be seen in Figure 6.6. The sudden and intensive use of the formerly rarely used RPC service can be clearly noted as a sign of anomalous host behaviour.
The very characteristic ICMP activity pattern of Nachi can be observed in Figure 6.7. It is striking that IP addresses containing the byte 0xC5 (=197 decimal) are not scanned, which results in black lines in our ICMP activity pattern.
activity plot (see white arrows in Figure 6.7). This can be explained according to an article in the Virus Bulletin [60] by the fact that for both of Nachi’s exploits, RPC DCOM and WebDAV, the worm 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. Scanning patterns involving many target networks can almost exclusively be observed within a larger network or a backbone.

6.4.2 Witty Worm

The Witty worm [114, 132] was first observed in the Internet on Saturday, March 20th, 2004, at approximately 4:45 UTC. It exploits a bug in several software products by Internet Security Systems ISS [75]. 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 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.

Compressibility Analysis

The Witty worm (or one having a similar entropy signature) can be detected by tracking the entropy of certain network traffic parameters like IP source address, IP destination address, source port, or destination port. Feinstein et al. describe in [54] a method for the detection of DDoS attacks, in which they estimate the entropy of such header values by using sampled probabilities of observed values. For worm outbreak detection, Arno Wagner developed an efficient compressibility based entropy estimation presented in [136], which we now briefly explain. A sequence of the same packet header parameter (e.g. IP address), over e.g. a 5 minute interval, is stored in an array. Then this array is compressed using the very fast lzo [98] compression algorithm and the ratio of compressed vs. original data size is calculated. This ratio is then plot-
Figure 6.8: The Witty outbreak clearly changes the compressibility ratio.

ted over time. In Figure 6.8, 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 destination ports and IP addresses become less compressible. This is due to randomly chosen destination ports and IP addresses in the Witty packets sent by infected hosts.

6.5 Discussion

Classical intrusion detection 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 chapter 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 chapter 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 chapter 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.

6.6 Conclusions and Outlook

Our open source framework UPFrame [111] 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.
Chapter 7

Host Behaviour Based Early Detection of Worm Outbreaks in Internet Backbones

7.1 Introduction

As we currently approach one billion Internet users [37], more and more cyber criminals join in and misuse this worldwide network by setting free malicious worm code that infects hosts and aggressively spreads over the network.

Based on the observation that hosts infected by the same worm execute the same code for scanning and transferring exploit and worm code, we assume that during a worm outbreak the network behaviour of many hosts will suddenly change in a similar way. In this chapter, we propose a novel near real-time method for early detection of worm outbreaks in
high-speed Internet backbones. By analysing backbone traffic at flow-level, we can attribute various behavioural properties to hosts like ratio of outgoing to incoming traffic, responsiveness and number of connections, which all are strongly influenced by a worm outbreak. These properties are used to group hosts into distinct classes according to their current behaviour. We show that by tracking the cardinality\(^1\) of these classes for significant changes over time, worm outbreak events can reliably be detected and a set of potentially infected hosts can be identified.

The outline of this chapter is as follows: After a survey of related work in Section 7.2 (Related Work), we present in Section 7.3 (Host Behaviour Based Worm Detection) our worm detection method. In Sections 7.4 (Validation on E-Mail Worms) and 7.5 (Validation on Network Worms), we validate our method on NetFlow traces from the DDoSVax traffic archive (see Chapter 4) that contain past real e-mail and network worm activity, and show cross-checks with traffic of “worm-free” Internet days. Finally, we draw our conclusions in Section 7.6 (Conclusions and Outlook).

7.2 Related Work

Intrusion detection for local area networks is a well established research area. In 1990, the Network Security Monitor [68] was one of the first intrusion detection tools that implemented the “connection counter” algorithm of the University of California in Davis. This algorithm is used to identify infected hosts and is based on the observation that worm infected hosts normally open connections at higher rates than uninfected ones. Monitoring bandwidth usage on network links and alerting upon reaching a threshold by statistical-based intrusion detection systems is similar to our traffic class cardinality tracking but less accurate.

Intrusion detection in backbone networks is a rather new research area. For worm detection in the global Internet, methods based on distributed intrusion detection systems like NetBait [35], firewall logs, or honeypots or a detection method using ICMP error messages [16] have been published. Most detection methods proposed for local area networks require packet payloads, which are expensive to collect and process in high-speed networks. Even analyses of real worms in backbones are extremely rare

\(^{1}\)“Cardinality” denotes the number of hosts in a single class.
due to the required large efforts of handling such data and due to privacy concerns. In the DDoSVax [134] project, we analysed major Internet worm outbreaks in the AS559 backbone on flow-level. Our analyses of the worms Blaster and Sobig.F can be found in Chapter 5. CAIDA analysed backscatter traffic of the Code Red, Slammer and Witty worms collected in a large unused IP address space with their Network Telescope [89]. Our proposed method is based on flow-level information of backbone routers and does not need packet payloads. Worm detection in backbones is a challenge: It has to be efficient for large traffic volumes, and there is no detailed information such as installed software about the active hosts, which is strongly relied on by many commercial intrusion detection systems for local area networks.

7.3 Host Behaviour Based Worm Detection

7.3.1 Method

From manual analyses of flow-level worm traffic, we gained insights into characteristic network traffic of worm outbreaks and worm activity. It seems that the classical client/server paradigm does no longer hold for Internet hosts and grouping hosts into clients and servers would be unwise. Popular P2P services, uploads of large files or the use of active FTP have intermixed client and server roles for Internet hosts. For our detection method, we define three core host behaviour classes “Traffic”, 

![Host Behaviour Classes Diagram]

**Figure 7.1: Host Behaviour Classes**
Class | Parameter | Threshold
--- | --- | ---
Traffic | (Bytes sent)/(Bytes received) | >3/min
Responder | bidirectional connections | >1/min
Connector | outgoing connections | >10/min

Table 7.1: Threshold conditions for host behaviour core classes

“Connector” and “Responder” that overlap each other as shown in Figure 7.1. The observed flow-level traffic of a host must satisfy a given threshold condition within a fixed time interval (1 minute in this chapter) in order to be member of a core class. We list the threshold parameters and the values used for all behaviour class plots in this chapter in Table 7.1. These classes were chosen such that a host’s membership in these classes is more likely if it was infected by a worm and less for hosts exhibiting “normal” network usage. The given parameter values proved useful for AS559 but might need adjustments for different backbones. During a worm outbreak, we expect a sudden and significant increase in the cardinality of one or several classes.

Hosts in the “Traffic” core class send several times more traffic than what they receive. This is typical for, e.g., worms that send out exploit code or that spread in e-mail attachments. Hosts in the “Responder” core class hold bidirectional connections. We define a bidirectional connection as a pair of flows in opposite directions between a pair of hosts, where the start time of the flows falls within an interval of less than 50 ms. Hosts that respond to TCP scans or handshake initiations typically have such bidirectional connections during a worm outbreak. Hosts in the “Connector” core class initiated many outgoing connections that are not necessarily bidirectional, which is typical for hosts that scan others.

The possible overlap of the core classes results in a total of eight distinct host behaviour classes that a host can be in: \( T(\text{Traffic}) \), \( R(\text{Responder}) \), \( C(\text{Connector}) \), \( T \cap C \), \( T \cap R \), \( R \cap C \), \( T \cap C \cap R \), and “no class”.

7.3.2 Implementation

To accommodate large volumes of NetFlow data, a filter stage that selects flows by protocol type (one or several of TCP, UDP or ICMP) and optionally by source and destination port numbers was prepended to...
our algorithm. We call it a near real-time algorithm as there are several small delays in the traffic processing path: network packets in the routers are aggregated to flow records, which are exported every four seconds, these flows are collected and preprocessed by our algorithm during a one minute interval and at the end of each interval, the class cardinalities are finally calculated.

The algorithm accepts incoming flow records passing the filter for the current and the next one minute interval. For each interval, a hash table stores the hosts seen together with the parameters for the 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. A 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 in the current interval. An efficient lookup by a hash key of destination IP address, source and destination port in this returned hash table is used to match a new flow with an existing one in the opposite direction.

The implementation of the algorithm was done in C as a plug-in for UPFrame (see Chapter 6 and [111]), our open source UDP packet processing framework that was designed for near real-time processing of bursty NetFlow data. To ensure robust operation during attacks, an upper plug-in memory limit was set. If more memory than the limit were needed, new flow records in the current interval are discarded and an error code for the interval is reported. When flows for the next time interval arrive, the algorithm resumes its operation. This ensures that the plug-in can also handle large-scale attacks without crashing. The memory consumption of the plug-in can be roughly estimated in bytes by\(^2\):

\[
\text{memory} = \#\text{active intervals} \cdot (\#\text{unique IP addresses per interval} \cdot 209 + \#\text{flows not discarded by protocol/port filter} \cdot 44).
\]

There is some additional management overhead of approx. 6 MByte. As an example, if we use two active intervals and consider 200,000 active unique hosts and 1 mill. flows per minute, in total 170 MByte of main memory are used. The plug-in can also detect missing data (e.g. due to an interruption of incoming NetFlow records) and will automatically

\(^2\text{We use symbol \# for “number of”}\).
start a new time interval if it suddenly receives NetFlow records that carry timestamps outside the currently observed two time intervals. The output of the plug-in is one binary statistic file of 6 KByte for each hour of NetFlow data processed that contains all class cardinalities and an operational state (i.e. missing data, memory limit reached etc.) for each of the 60 minutes.

7.4 Validation on E-Mail Worms

In this section, we analyse the detection effectiveness of our algorithm on two major Internet e-mail worms, namely Sobig.F (2003) and MyDoom.A (2004). We applied our host behaviour algorithm to SMTP traffic, i.e. we considered only flows to and from port 25/TCP.

7.4.1 Sobig.F

On August 19th, 2003, around 9:00 UTC [46]), the Sobig.F e-mail worm (see Chapter 5.3 and [31]) that runs on Microsoft Windows 95 or higher first appeared in the Internet. Besides spreading via executable e-mail attachments of varying sizes and providing its own MTA for sending infected e-mails, the worm is programmed to update itself at predefined dates by downloading new code from predefined computers. By timely intervention of network operators and system administrators this update mechanism could be blocked by shutting down all 20 predefined servers. The original worm was programmed to disable itself on September 10th, 2003. Date and time are taken from a small set of hard-coded global NTP time servers. The e-mails sent use an arbitrary sender and recipient address taken from local files of the infected host.

The graph in Fig. 7.2 shows the total number of bytes per hour transferred as e-mail (SMTP) traffic over the SWITCH border routers. The daily rhythm can be clearly seen. The workdays have traffic with a maximum around 5 GByte/hour, on Saturdays and Sundays the traffic is considerably less. On a workday, the lunch break can easily be identified. The peak of the Sobig.F outbreak is reached on Tuesday, August 19th, 2003 in the hour of 12:00-13:00 UTC with 21.7 GByte/hour, which is almost a fivefold increase compared to “normal”.

By filtering the flows for SMTP traffic (i.e. to or from 25/TCP) and
tracking the cardinality of the host behaviour classes $T \cap C$ in Figure 7.3 and $C \cap R$ in Figure 7.4, the outbreak could have been detected early around 10:00 UTC by comparing with traffic two weeks before (see lower plots in the figures) or by cardinality threshold alerts. A fast reaction (e.g., timely information to users about the new e-mail worm and e.g., traffic filter rules to block new SMTP senders) could have strongly mitigated the impact.
Figure 7.3: Cardinalities of the “Traffic ∩ Connector” class two weeks before and during Sobig.F
Figure 7.4: Cardinalities of the “Connector ∩ Responder” class two weeks before and during Sobig.F
7.4.2 MyDoom.A

On January 27th, 2004, the MyDoom.A [32] (a.k.a. W32/Novarg.A) e-mail worm began to massively spread worldwide. It spreads via executable e-mail attachments and provides its own MTA for sending infected e-mails. It uses various different attachment file extensions such as .scr, .exe, .pif, .cmd, .bat, or .zip. An arbitrary recipient and sender are taken from e-mail addresses found on the infected host. The recipient must explicitly open the malicious attachment to infect the system. Additional "features" of this worm are that it copies itself to the local KaZaA P2P file sharing directory if available and that it starts a DDoS attack on www.sco.com after a reboot on Feb. 1st 16:09:18 UTC. After Feb. 12th, this attack ceases. The worm also installs a backdoor on ports 3127-3198/TCP, which remains active.

The graph in Fig. 7.5 shows the total number of bytes per hour transferred as e-mail (SMTP) traffic over all SWITCH border routers. The daily rhythm can clearly be seen. The two workdays in the graph show e-mail traffic with a maximum around 11 GByte/hour, whereas on Saturday and Sunday the traffic is considerably less as can be easily explained by the absence of business related e-mails. On a workday, the lunch break can easily be identified. The additional MyDoom.A e-mail traffic elevated the traffic volume level of e-mails since early on Tuesday 27th. Its spreading became evident in the traffic volume statistic just after lunch time as indicated with an arrow in the graph.

During the spreading of MyDoom.A, we observed an increase of 14%
(for Tue, 27th Jan. 2004) to 30% (for Wed, 28th Jan. 2004) in e-mail traffic volume in bytes per hour. Compared to the e-mail worm Sobig.F of August 2003, which caused an almost fivefold increase in e-mail traffic load within the initial hour of its spreading, the MyDoom.A worm seems to be quite harmless.

A prevalent behavioural pattern of MyDoom.A is that new hosts become active by sending e-mails and that existing hosts (especially network address translation (NAT) gateways) send massively more e-mails. This

Figure 7.6: Cardinality of the $T \cap C$ and $T \cap C \cap R$ classes before and during MyDoom.A
fact can clearly be seen in our behaviour class analysis in Figure 7.6. Comparing the cardinality of host behaviour classes to just before the outbreak or to the same days of the previous week (lower graph in Figure 7.6), the outbreak becomes evident. In contrast to the volume statistic, in which the outbreak becomes evident around 11:00 UTC, the outbreak could have been noticed already around 8:00 UTC if such an early worm detection tool had been in place. The sudden drop in the cardinality of the observed classes in the afternoon was due to manually installed SMTP traffic blocking filters in some networks that were strongly affected by this worm.

The number of e-mails sent (i.e. SMTP flows) also shows an irregular behaviour as can be seen in Figure 7.7. The number of source hosts sending out e-mails over SMTP as plotted in Figure 7.8 increases notably as well due to MyDoom.A.

Figure 7.7: MyDoom.A worm: SMTP flows

Possible reasons for the rather moderate e-mail worm traffic volume increase during MyDoom.A in comparison to Sobig.F’s explosive growth of traffic volume are:

- **Smaller worm size**: The MyDoom.A with a length of 22 KByte is considerably smaller than Sobig.F with about 71 KByte. Hence it would need 3.2 transmissions of the MyDoom.A worm for a single Sobig.F transmission and hence MyDoom.A does not show up as prevalent in bandwidth usage statistics as Sobig.F did.

- **Public awareness of e-mail worms**: Many computer users did
not activate the worm code by executing the attachment as they possibly still remembered Sobig.F.

- **Up-to-date anti-virus software**: Novarg/MyDoom virus signatures for e-mail virus scanning software were available by late Jan. 27th, 2004 from most anti-virus software vendors.
- **Firewall SMTP filtering**: Corporate network operators installed egress filtering to catch e-mails sent from their client computers to arbitrary Internet e-mail servers to counter the Sobig.F attack, which also prevented the direct delivery of MyDoom.A generated e-mail to possible victims.

As seen, our method can effectively and reliably detect fast spreading e-mail worms in the early stage of their outbreak by tracking the $T \cap C$, $C \cap R$ and possibly also the $T \cap C \cap R$ behaviour classes for significant changes. We presented only the classes with the most significant changes in the plots. Watching merely the cardinalities of any of the three core classes $T$, $C$ or $R$ would not be sufficient.
7.5 Validation on Network Worms

In this section, we validate our method on fast spreading network worms, namely Witty and Blaster. Our algorithm does not need previous knowledge about the specific ports used or the type of the exploited vulnerability in order to detect such worms. We merely need to track the cardinality of the classes for UDP traffic (Witty) or UDP and TCP traffic (Blaster) for detecting the outbreaks. No prepended port filter was used to detect these worms.

7.5.1 Witty Worm

The Witty worm [132] was first observed in the Internet on Friday, March 20th, 2004, at approximately 4:45 UTC. It exploits a bug in several products by Internet Security Systems (ISS [75]). Witty uses UDP packets with source port 4000 and a randomised target port. The vulnerable host population for Witty was approximately 12,000 hosts. Witty was the first fast worm which demonstrated that even a small vulnerable population can be infected in a few hours. Unlike most other worms, it carried a destructive payload for deleting arbitrary blocks on an infected computer’s hard disks.

Hosts infected by Witty created many outgoing UDP flows (“connections”) and show up clearly in the Connector class plot in Figure 7.9. Around noon, it seems that for almost three hours a filter was installed, which was later deactivated again. By comparing the Connector class cardinality of the “Witty day” with the regular “Witty-free” Saturday one week before (lower graph in Figure 7.9) the worm outbreak can be seen even clearer.

7.5.2 Blaster Worm

The Blaster.A worm (see Chapter 5.2 and [30]) was first observed in the Internet on August 11th, 2003, with a massive outbreak starting 16:35 UTC [46]. The worm exploited a remote procedure call (RPC) vulnerability of Microsoft Windows 2000 and Windows XP operating systems that was made public in July 2003 [129] and is described in the Microsoft Security Bulletin MS03-026 [88] as critical. Estimates of Blaster infected hosts range from 200,000 [2] to 8 millions [84].
The host behaviour method notices Blaster activity by a huge increase of approximately 4,500 in the number of hosts responding per minute and a higher fluctuation rate (considering any TCP and UDP flows). Previously silent and client-like hosts suddenly start to respond to connection requests. This is due to infection attempts by Blaster, mostly on port 135/TCP. We attribute the short peaks in the number of responders before the Blaster outbreak to TCP scanning activities. A “Connector” class analysis of Blaster revealed a limited spreading Pre-Blaster variant in the late afternoon on August 10th of 250 additional hosts, 160 of which also connected to 69/UDP that was used by Blaster for worm code.
Figure 7.10: “Responder” class cardinality before and during Blaster.
transfer. The cardinality of the Traffic class or of hosts in a combination of classes did not show significant changes during the Blaster outbreak.

7.6 Conclusions and Outlook

We presented a generic near real-time method for early detection of worm outbreaks in high-speed Internet backbones. By classifying the network behaviour of Internet hosts in a way that is highly sensitive to newly infected hosts, and by tracking the cardinalities of these classes and alarming on sudden and rapid increases, we can warn early of worm outbreaks. We successfully validated our method on real flow-level backbone traffic of four major past Internet worms. We showed that our method works well for e-mail based worms like Sobig.F and MyDoom.A as well as for fast spreading network worms like Witty and Blaster.

The trend to hide Internet hosts behind network address translation (NAT) gateways and to use proxies to access Internet services will result in fewer hosts (i.e. source IP addresses) showing anomalous behaviour visible in the backbone. This might require an extension of this approach that identifies, observes and analyses such gateways and proxies in more detail. Future work will also be to establish automatic alarming mechanisms based on class cardinality tracking and to adjust them for an optimal security incident sensitivity in the observed network.
Early detection of attacks will remain a hot research topic for the future. The earlier and more reliably newly started Internet attacks with potentially devastating effects to network services can be detected, the higher the quality of service in the Internet will be.

The idea of the DDoSVax project to continuously capture border router backbone traffic for several years in the hope to capture massive Internet attacks for later analysis proved fruitful. The DDoSVax flow-level archive is very comprehensive and contains all Internet attacks that hit AS559 since early 2003 like e.g. Blaster, Sobig.F, MyDoom, Witty, or Sasser. It would be possible to also use this archive for research that is not security related like e.g. for traffic engineering or to extract Internet service usage trends over several years.

Our forensic analyses of the network worm Blaster and the e-mail worm Sobig.F revealed many interesting findings and yielded some lessons learnt for network operators. We are convinced that from analysing other major attacks in detail, an even broader basis could be laid for
building new detection methods and judging the effectiveness of defence strategies used. Each major worm seen so far was different in its prevalent traffic characteristics. The challenge lies in finding further generic approaches to detect such unwanted attack traffic reliably.

Our framework UPFrame helped us to test our near real-time online detection algorithms as plug-ins concurrently on the same input data. It considerably reduced programming complexity and increased development efficiency. The presented detection plug-ins were found to be sensitive to massive attacks and provide a valid basis to build a backbone monitoring and early warning system.

Finally, our host behaviour based worm detection approach turned out to be very generic in the sense that it not only makes network worms like Blaster and Witty clearly visible, but also E-Mail based worms like Sobig.F and MyDoom.

Next steps to build an early warning system from our detection methods presented would be to find approaches for (semi-)automatically setting reasonable static and adaptive warning thresholds and to find further signatures for known worms, attacks and anomalies as well as for new types of worms and attacks not yet seen in the Internet.
Part III

Mitigation
Chapter 9

Enhanced Internet Security by a Distributed Traffic Control Service Based on Traffic Ownership

9.1 Introduction

Recent massive Internet worm outbreaks such as Slammer [90], Blaster [127] or Sasser [128] have shown that a large number of hosts that goes into the millions [84] are patched lazily or are operated by security-unaware users. Such hosts can be compromised within a short time and misused to run arbitrary and potentially malicious attack code transported in a worm or virus or injected through installed backdoors. Dis-
Distributed denial of service attacks (DDoS) use such poorly secured hosts as attack platform and cause degradation and interruption of Internet services, which result in major financial losses, especially if commercial servers are affected [47]. In recent years, such attacks were repeatedly used for blackmailing companies offering casino, sport bet or advertising distribution [69] services on the Internet. The attacks' structures differ, but all aim at rendering a service unavailable for legitimate clients. A large number of malicious hosts sends unsolicited network traffic and hereby exhausts network or host resources.

Keeping a commercial server up and running 24/7 is an asymmetric struggle: while attackers are able to exploit the processing and bandwidth resources and the flexibility of a huge number of compromised hosts to install malicious tools and launch new attack variants, operators of Internet servers are left without appropriate means to counteract attacks. Widespread availability of attack tools makes it easy for non-experts (i.e. script kiddies) to carry out even large-scale attacks. As a consequence, new attacks appear frequently, while defence strategies lag far behind. We believe that current security technologies and concepts that focus on end system and access networks soon cannot cope anymore with the growing number and the increasing intensity of Internet attacks. We are convinced that large-scale attacks can only be efficiently handled by providing increased security within the network.

In this chapter, we present a novel distributed traffic control service, which can help to enhance Internet security significantly. At its core is a safe delegation of network management capabilities. The traffic control service is based on adaptive network traffic processing devices that can be deployed incrementally in the Internet close to routers. As one specific application domain, we show how such a service can fight DDoS reflector attacks, which are tracked down unsatisfactorily and in some cases are handled even counterproductively by existing security mechanisms. Our service can help to stop attack traffic within the network as close to the Internet uplink of an attacker as possible. Our adaptive traffic control service is in no way limited to security related applications. It also enables many other new applications such as for example new ways of collecting traffic statistics, distributed network debugging, inter-domain service level agreement validation and support for forensic analyses.

The chapter is organised as follows: In Section 9.2, we present DDoS attack scenarios. In Section 9.3, we analyse various mitigation mecha-
nisms and show the ineffectiveness of several proposed techniques and systems against DDoS attacks. In Section 9.4, we propose our new traffic control service based on the fundamental concept of “traffic ownership” and adaptive traffic processing devices. The infrastructure we rely on for our service is explained in Section 9.5. Section 9.6 presents a use case describing in detail how our system can be applied to mitigate DDoS attacks. In Section 9.7, we draw our conclusions and give an outlook on future work.

9.2 Attack Scenarios

9.2.1 Distributed Denial of Service Attacks

In an Internet DDoS attack, compromised hosts of security unaware users are usually remotely controlled and organised by an attacker as a so called amplifying network of masters and agents. They are then misused to carry out attacks on a few or just a single host. Such attacks can also be targeted at core Internet infrastructure components such as routers, central services (e.g. domain name system) or low to medium bandwidth links.

The common aim of DDoS attacks is to deny certain services or resources to prospective users. A large diversity of attack forms exists in the Internet. In [116] a taxonomy of denial of service attacks in the context of networks is presented. Technically, a partial or complete denial of service can be caused by exploiting a system weakness to make a specific host crash, by exhausting a host’s computational, storage, memory or other resources with the initiation of expensive calculations (e.g. public key de-/encryption) or by triggering resource consuming operations (e.g. complex database queries) or using up all disk space (e.g. by making a host write huge log files). Other ways to cause denial of service are the misuse of protocols that make the victim host seem to be temporarily unavailable due to faked protocol signalling (e.g. sending ICMP unreachable messages or TCP reset packets) or the very commonly used technique of flooding a target router, host or network link with a huge number of packets at fast rates such that many packet losses occur and legitimate traffic is hindered from reaching its destination. The many forms in which DDoS attacks occur in todays Internet make it highly
nontrivial to find a panacea for mitigating or stopping such attacks. Attackers can make use of Internet worms as it was done with e.g. MyDoom [131] for compromising hosts and installing a backdoor. This allows to build up a huge amplifying network of several ten thousand hosts in a short time.

9.2.2 DDoS Reflector Attacks

A rather new variant of DDoS attacks became known as DDoS “reflector” attack. This attack form is especially difficult to defend against as the victim is flooded with traffic from ordinary Internet servers that were not even compromised. Gibson gives details in [62] about the reflector DDoS attack that blasted the grc.com site off the Internet in January 2002.

![Diagram of DDoS reflector attack setup](image)

Figure 9.1: A generic DDoS reflector attack setup

A selection of DDoS reflector attacks is described in [103]. Any server that supports a protocol which replies with a packet after it has received a request packet can be misused as a reflector without the need for a
server compromise. Some prominent examples are web servers, Gnutella servers that even initiate new connections on behalf of other hosts, FTP servers, DNS servers and routers. They return SYN ACKs or RSTs in response to TCP SYN requests and other TCP packets, or ICMP “time exceeded” or ICMP “host unreachable” messages as a reaction to certain IP packets.

Figure 9.1 shows that the agents send their packets with the spoofed source address set to the victim’s address (V) to “innocent” servers with IP addresses Ri. These servers act as reflectors. The source addresses of the actual attack packets (Ri) received by the victim are not spoofed. They belong to legitimate uncompromised servers. Stopping traffic from these sources will also terminate access to Internet services that the victim might rely on.

DDoS attacks organise master and agent hosts in the way of an amplifying network as depicted in Figure 9.1. A quite popular choice for the communication channels between the masters and the agents is to misuse an Internet Relay Chat (IRC) system with weak user authentication. Such a network amplifies the rate of packets (a few control packets of the attacker to the masters cause many attack packets sent by the agents to the victim), the size of packets (if request packet size < reply packet size) and the difficulty to trace back an attack to the initiating attacker. We will come back to these core properties when discussing security aspects of our traffic control service.

9.3 Analysis of Mitigation Mechanisms

This section presents related work that addresses mitigation strategies against DDoS attacks. We distinguish two basic mitigation schemes, reactive and proactive, which are analysed in more detail and discussed with regard to their mitigation effectiveness and implementation complexity. We show that earlier proposed mitigation schemes fall short of counteracting certain classes of DDoS attacks. In some cases mitigation schemes even amplify the effects of an attack as legitimate servers or complete networks are cut off from the network.
9.3.1 Reactive Mitigation Strategies

Reactive schemes often proceed in three phases. In the first phase, distributed monitoring components try to detect on-going DDoS attacks. Once an attack is detected, the detector triggers the second phase that aims at locating the attack sources. In the third phase, countermeasures are deployed to mitigate the attack. After the attack has finally stopped, countermeasures have to be relieved or removed.

A lot of prior work concentrated on the second phase by tracing back packets with spoofed source addresses to their actual origin [109, 118, 121, 139]. While it is very valuable in forensics to find the origins of the attack, this deals with neither detecting attacks nor deploying any dispositions against ongoing attacks. Traceback mechanisms play an important role in other reactive mitigation schemes to determine where countermeasures should be deployed and which filtering rules should be applied. If DDoS attacks involve reflectors, traceback mechanisms will yield a wrong “attack source” – the reflectors – to be identified and possibly filtered. Hence, access to important services might be blocked because DNS or web servers are often abused as reflectors. In fact, with todays Internet infrastructure it is almost impossible to effectively defend against reflector attacks without either extending the majority of Internet servers with new logging facilities or by deploying a worldwide packet traceback mechanism, many of which were suggested in the literature but are rarely used in practice.

The authors of [81] propose that attacked hosts set filter rules to limit the incoming traffic at the last hop IP router. The network infrastructure is assumed to be able to deal with traffic bursts, while the attacked host is not able to process incoming traffic. An open question is, whether a host is still able to configure filter rules while its resources are exhausted under a DDoS attack. Moreover, the authors of [81] propose a DDoS defence mechanism based on the Internet Indirection Infrastructure (i3) [124]. i3 is implemented as an overlay network that is used to route a client’s packets to a trigger and from there to the server. Due to performance concerns, i3 would only be used if a server were under attack. Otherwise, communication would be established directly between client and server. In order to use i3 as a defence mechanism, IP addresses of the attacked servers are assumed to be hidden from the attackers. It remains unclear how server IP addresses can be hidden under attack, when they are
known under normal operation. Pushback [85] performs monitoring by observing packet drop statistics in individual routers. Once a link becomes overloaded to a certain degree, the Pushback logic, which is co-located with routers, classifies dropped packets according to source addresses. The class of source addresses with the highest dropped packet count is then considered to originate from the attacker. Filter rules to rate limit packets from the identified source address(es) are automatically installed on the routers on the path towards the source(s) of attack. Routers on the attack path are informed about the detected attacks and install the same rules. In this way, the attack is pushed back and confined. Pushback assumes that DDoS attacks result in overloaded links. In many cases, however, an attacked server’s resources are exhausted before its uplink is overloaded. In particular, this is the case for servers that are hosted in farms, where the communication link is provisioned to feed a large number of servers. Moreover, rate limiting flows based on source addresses is not adequate if addresses are spoofed. In this case, legitimate sources may experience severe service degradation. The Pushback protocol [58] requires all routers on the attack paths to collaborate. If a router on a path between attacker(s) and victim does not speak the protocol, the Pushback of filter rules stops to extend further on that particular path. An inherent problem of reactive mechanisms is that it is very difficult to detect DDoS attacks. None of the discussed systems with the exception of Pushback addresses this issue.

9.3.2 Proactive Mitigation Strategies

Proactive strategies intend to reduce the possibility of successful DDoS attacks by taking appropriate provisions prior to attacks. Ingress filtering [57] rejects packets with a spoofed source address at the ingress of a network (e.g. to the Internet service provider’s backbone network). As spoofed source addresses are used in several attacks, this approach when put into widespread operation renders many attacks inefficient. Attacks involving reflectors with legitimate source addresses, however, are only affected if ingress routing is applied on paths between agents and reflectors (see Figure 9.1). Performing ingress filtering puts a management burden on ISPs because they must keep all filtering rules up to date and defective rules will disgruntle their customers. Even though
Ingress filtering was already proposed in 1998 [56] to prevent attacks, it was incompletely applied worldwide as current attacks show. The recent Spoofer project [17] by MIT found that about one quarter of their 150 autonomous systems (AS), 70 mill. IP addresses and 229 network blocks tested allowed full or partial spoofing using UDP packets. Scaling these results up to the whole Internet yields over 360 mill. IP addresses and over 4,600 ASes from which spoofing is possible.

Secure overlay networks such as SOS [77] and Mayday [8] require each communicating user of a group to pre-establish a trust relationship with the other group members. Hence, a user may be required to participate in many groups. As management of many trust relationships is costly and potentially large amounts of traffic is routed among overlay nodes, overlay-based proactive solutions are not adequate for communication with popular web servers (e.g. Google, CNN, amazon, ebay, etc.), which include millions of communicating hosts. Furthermore, keeping malicious users out of an overlay will be a challenge for a large user base.

9.3.3 Discussion of Mitigation Effectiveness

We have seen that the described reactive mitigation schemes fail to be effective against DDoS attacks in all three phases: attack detection, attack location (traceback) and attack mitigation (filtering). What makes DDoS attacks so hard to come by is the fact that attack traffic generally contains spoofed source addresses. In DDoS reflector attacks this is even more complex because the victim does not receive traffic from the DDoS agents directly, but from legitimate sources without spoofed source addresses. If source spoofing were impossible, reflector attacks could be completely prevented as the agents could no longer spoof the victim's IP address for using it as source of their attack packets sent to the reflectors. Furthermore, complex traceback mechanisms would not be needed because the originator of malicious packets could directly be identified by the source address in those packets.

Proactive approaches may be implemented directly in the IP network or as an overlay network. An advantage of overlay-based solutions is that they can be deployed incrementally, without requiring the cooperation of ISPs. Users only participate in a secure overlay if the risk of DDoS attacks against them and resulting costs exceed their effort to participate in the overlay.
9.4 Distributed Traffic Control by IP Address Owners

More effective defence strategies are possible within the IP network. Performing ingress filtering, a single router is capable of blocking traffic from a large number of malicious nodes. As explained in Chapter 9.5.4, ingress filtering combined with route-based packet filtering is highly effective against source address spoofing even if only approximately 20% of the autonomous systems have it in place. As a consequence, the network itself should offer appropriate means for defence. Defence mechanisms must be implemented by the Internet service providers (ISP) and backbone service providers (BSP) because they control the traffic entering their network and have access to technology that allows them to deal with large volumes of traffic. However, ISPs and BSPs currently lack any incentive to implement mechanisms that protect network users from attacks.

9.4 Distributed Traffic Control by IP Address Owners

Today's Internet is controlled by network operators, namely Internet and backbone service providers. Network users are restricted to control traffic at their Internet uplink and cannot manage or control network traffic within the Internet.

9.4.1 Network Traffic Control Service

We propose a novel service that enables network operators to safely delegate specific traffic control features to network users. That for, we innovated the fundamental concept of traffic ownership [44]. We declare a network packet to be owned by these network users, who are officially registered to hold either the destination or the source IP address or both of that packet. The delegation of certain network management capabilities from network operators to network users is safe in the way that our system assures that a network user can only get control over the IP packets he or she owns. By adding even further restrictions on the traffic control capabilities as discussed in Section 9.4.5, we can prevent misuse and malicious interference with other traffic. If the source and the destination address of a network packet belong to different parties then this packet can be controlled subsequently by two different parties.
For practical reasons, traffic control can be executed by a designated party on behalf of a network address owner.

![Figure 9.2: Router extension with traffic processing device (TPD)](image)

Our system consists of remotely programmable network traffic processing devices as shown in Figure 9.2. The owner of a network address or range gets access to the management of some or all of these devices after having registered for the distributed traffic control service. Traffic entering a router is redirected to a nearby adaptive traffic processing device (TPD) only if it carries an IP address as source or destination, which the TPD was set up for (see Section 9.5). Most traffic will use the direct “IP fast path” through the router.

When the TPD processes a network packet, it first executes traffic control on behalf of the owner of the source IP address. Subsequently, it executes traffic control on behalf of the owner of the IP destination address. This is analogous to the high-level communication process of first sending an Internet packet by the source (and hence under its control) and then receiving it by the destination (and consequently under the recipient’s control). This control hand-over is performed at each TPD on the network path of an IP packet, which was activated for the source and/or destination IP address contained in the IP packet.

### 9.4.2 Traffic Processing Device Functionality

Our traffic processing devices are adaptive, which means that the functionality of the device can be extended and modified by installing new software (or hardware) modules when new demands arise. Furthermore, upon routing updates, the configuration of modules that depend on the topology can be either automatically adapted or they can be temporarily disabled. In the context of DDoS attack mitigation, we think of firewall-like services such as anti-spoofing filtering, packet dropping, payload
9.4 Distributed Traffic Control by IP Address Owners

deletion, source IP blacklisting or traffic rate limiting. Rules that match traffic by header fields, payload (or payload hashes), or timing characteristics etc. can be installed, configured and activated instantly. During attacks, triggers can activate predefined additional configurations.

To make such a distributed firewall even more powerful, each such device provides contextual information depending on where it is attached to the network and which traffic (stub, transit etc.) it can observe. Additionally, if made available by the network operator, the router’s state and configuration (e.g. static routing information, packet drop rates, congestion parameters, traffic mix, router load, flow-level statistics etc.) can also be provided. We can e.g. only prevent source spoofing effectively, if the adaptive traffic processing device is aware of whether it processes transit traffic of autonomous systems or only traffic from customers of a peripheral\(^1\) ISP.

9.4.3 Attack Prevention and Defence

For stopping a DDoS reflector attack to a specific web site, the owner of that web site’s IP address can, by using our proposed traffic control system, almost instantly deploy worldwide ingress filtering rules. These rules will block all traffic that enters the Internet from customers of a peripheral ISP and that carries this web site’s spoofed IP address in the packets. This will very effectively stop IP packets from DDoS agents to reflectors carrying illegitimately the victim’s IP as source address. Of course, legitimate transit traffic, the traffic of the peripheral ISP, where this web site is attached to, and traffic to clients located at peripheral ISPs must not be blocked, as we want the web site’s reply packets to reach the legitimate hosts requesting service from it. The more ISPs offer such a distributed traffic control service, the more effective such a defence will be. Our service allows for filtering traffic close to the source of the attack. Hence, we can heavily reduce collateral damage caused by compromised hosts acting as attack agents\(^2\). Whereas ingress filtering itself is not new, the way we allow network users to remotely deploy such filtering specifically for their IP addresses is novel.

---

\(^1\)A peripheral ISP operates stub network(s) and transports no transit traffic.

\(^2\)Our ISP based ingress filtering strategy cannot prevent an IP spoofing attack originating from the same enterprise network domain as the attacked service resides in. However, enterprise firewalls can provide solutions for this case.
Attacks based on protocol misuse, such as sending ICMP “host unreachable” or TCP RST messages to tear down TCP connections can also be filtered out. Without a distributed traffic control service as provided by our system, worldwide filtering of illegitimate packets is almost impossible due to the many network operators involved that have to be contacted individually for setting up filter rules all over the globe.

9.4.4 Emerging Applications

Our distributed traffic control service is in no way limited to firewall-like functionality as new software and hardware modules can be installed on the traffic processing devices when needed. Other services can be based on logging data, collecting traffic statistics or triggering events.

To illustrate the multipurpose nature of our infrastructure, we briefly describe additional use cases of our system.

**Traceback:** Our system could be used to implement a worldwide packet traceback service such as SPIE [118] by storing a backlog of packet hashes. This would enable support for network forensics by sampling traces of suspicious network activity. Such a service would allow the network user to investigate the origin of spoofed network traffic. As explained in Chapter 9.5.4, ingress and route-based filtering can also provide a traceback service that limits the possible attack sources to a small number of autonomous systems.

**Automated reaction to network anomalies:** Our system allows placing triggers in the network in a distributed manner. Triggers generate events if a specific condition is met and thus can be used to signal the activation of a traffic filter function. Automated reaction to network anomalies could be implemented by placing triggers that fire an event if the traffic statistics (e.g. rate of connection attempts from/to a particular server) indicate values exceeding expected boundaries. As a consequence, a rule that rate limits the anomalous traffic could be activated.

**Network debugging and optimisation:** Our system provides means to collect traffic statistics within the network. Link delays or packet loss on intermediate links could be measured for network debugging purposes. As an example, such information could help providers of content distribution services to optimise their (overlay) network.

**Service Level Agreement Validation:** The validation of inter-
domain service level agreements (SLA) is a challenge as it involves multiple administrative domains. Our traffic control system provides a homogeneous service platform spanning multiple network operators. In addition, our traffic ownership restriction eliminates privacy concerns as third party network data, which is not required for SLA validation, is inaccessible. Our system is well suited to measure e.g. jitter, delay or packet loss on a hop-to-hop and end-to-end basis for validating SLAs. More on inter-domain SLA validation can be found in our paper [22].

9.4.5 Security Considerations

For the proposed distributed traffic control service to be accepted by network operators (namely ISPs and BSPs), it is vital, that such a device keeps the network manageable by the network operators and that it cannot be misused for an attack itself. This is addressed by the core of our approach, the novel concept of traffic ownership: We restrict the traffic control for each network address owner to his/her own traffic, i.e. packets to and from owned IP addresses. This allows our service to assure that traffic owned by other parties is not affected. In addition, data privacy concerns are eliminated as no third party traffic can be controlled or observed.

However, while this traffic ownership restriction is the most important one to ensure security and acceptance of our system, it alone does not yield a secure system. To prevent collateral damage caused by misconfigurations or malicious behaviour of users having access to such devices, we need to restrict traffic control even further. We do not allow the adaptive traffic processing device to modify the source and the destination IP address of a packet. Such rerouting could wreak havoc easily by causing routing loops or interference with other routing mechanisms, by allowing transparent source spoofing via changing the source address of an attack packet to a random one on a TPD close to the victim host, or by “forwarding” attack traffic by setting a new destination address in attack packets. Also the time to live (TTL) field of IP packets is a field that we cannot allow to be modified as it aims to set an upper bound of network resources a packet is able to use.

Furthermore, we need to prevent that the service can cause amplifying network-like effects as discussed in Section 9.2. The traffic control must not allow the packet rate to increase. In addition, the amount of the
network traffic leaving the traffic processing device must be equal or less\(^3\) compared to the amount of traffic entering it. This means that packet size may only stay the same or become smaller. To ensure these restrictions, new service modules for a traffic processing device must be checked for security compliance before deployment. Consequently, the danger of delegating partial control of the network from the network operator to the customers is very limited as countermeasures against effects of misconfigurations and misuse were taken into consideration when designing the distributed traffic control service. In fact, the above mentioned restrictions assure that traffic owned by other parties is not negatively affected by dynamically deployed service functionality and that network operators do not lose control over their network albeit their customers’ extended traffic control realm.

9.4.6 Incentives for Deployment

As a reaction to a DDoS attack on its web servers in February 2005, a German publisher posted a 10,000 EUR reward for hints about the identity of the originator of the attack [71]. Although this publisher’s business model is based on selling hard copies of journals and not on the news it provides (for free) on the web, the reward shows how much companies value an uninterrupted web presence. For other companies with a business model that depends on a web presence, our traffic control service is even more valuable. In mid-August 2005, the German company Fluxx offered 40,000 EUR [70] reward for hints that lead to the arrest of the hackers that were flooding their online betting office jaxx.de with a DDoS attack for several days and that blackmailed them by the same amount for stopping it. Hence, we see many incentives for ISPs and BSPs to deploy such a distributed traffic control system. It can be offered as a new premium service to customers that need to protect their commercial Internet servers from attacks, or that want to gather distributed traffic statistics for their sites. Besides using it for new security services, there are many other possible applications such as logging data, collecting traffic statistics, or validating service level agreements. Malicious or illegitimate traffic can now be filtered closer to the source. This frees valuable bandwidth resources and makes them available for\(^3\)For e.g. logging, statistics or trigger event services, we will allow a reasonable amount of additional traffic.
transporting legitimate traffic. Collateral damage is limited mostly to poorly managed access networks where infected or compromised machines are hooked up to the Internet. This is because attack traffic can be filtered by the new traffic control service at the uplink of such an ISP to a more security-aware ISP or BSP. Other advantages are that ISPs can offer new services and generate additional revenue, whereas customers of an ISP get better service and, e.g. can rapidly reconfigure the traffic processing devices in the network to their needs.

9.5 Infrastructure

This section describes the deployment of our traffic control service and its underlying network infrastructure.

9.5.1 Network Model

Our network model shown in Figure 9.3 distinguishes four different roles: Internet number authority, traffic control service provider (TCSP), Internet service provider (ISP), and network user. This section subsumes both types of organisations, ISPs and BSPs, under the role ISP. The TCSP manages the new traffic control (TC) service. It sets up contracts
with many ISPs that subsequently attach adaptive traffic processing devices to some or all of their routers and enable their network management system to program and configure these devices.

A network user must first register with the TCSP before using the traffic control service as shown in Figure 9.4(a). After checking the identity of the network user, the TCSP verifies the claimed ownership of IP addresses, which the network user wants to control traffic for. Therefore, it checks with Internet number authorities whether the IP addresses are indeed owned by the service requester. If everything is fine, access to the traffic control service is granted. The binding of a network user to the set of IP addresses owned and the subsequent verification when using the traffic control service (TC service) could be implemented with digital certificates signed by the TCSP.

After successful registration for the basic TC service, a network user may initiate the deployment of a specific service (e.g., ingress filtering), which is implemented on top of the TC service, see Figure 9.4(b). The network user requests the TCSP to deploy the specific service in the network. The network user may scope the deployment according to different criteria (e.g., “only on incoming links of border routers placed in stub networks”). The TCSP maps the request to service components and instructs network management systems of appropriate ISP’s to deploy and configure the service components. ISPs in turn deploy and configure the components on adequate traffic processing devices (TPD), see Figure 9.3, and configure their routers accordingly. Once the service is deployed, a network user may activate, modify specific parameters or read logs of the service. Therefore she sends corresponding requests to the TCSP, which relays them to the appropriate network management systems of the concerned ISPs.

Our infrastructure offers an alternative way to activate, modify specific parameters or read logs of the service. A network user may directly interact with the ISPs’ network management systems to control the processing of packets that contain a source or destination IP address she owns. For an efficient configuration of many traffic processing devices

---

4To check the network user’s identity the TCSP performs similar actions as a digital certification authority (CA), e.g. offline verification of an official identity card or online verification of a digital certificate issued by a trusted CA.

5Ownership of IP addresses is maintained in databases of organisations such as ARIN, RIPE NCC, etc.
(a) Service registration

(b) Service deployment

Figure 9.4: Message sequence diagrams
and routers, an ISP’s network management system can forward requested configurations to other ISPs’ network management systems upon request of the network user. This approach is particularly useful if the network conditions are such that the TCSP can no longer be reached, e.g. because of an ongoing DDoS attack on the TCSP. In principle, each ISP could establish a mini-TCSP and offer the traffic control service limited to his network. However, this would make worldwide deployment of traffic control based services cumbersome. The introduction of a TCSP helps to scale the management of our service. Only a single service registration is needed instead of a separate one with each ISP.

The infrastructure can be deployed incrementally. Most traffic control based services will be useful even if not all ISPs offer it. They become more effective when more ISPs join. For example, anti-spoofing protection and firewall-like services can filter closer to the source and therefore less network resources will be wasted.

9.5.2 Traffic Control Unit Architecture

A Traffic Control Unit (TCU) is the combination of one or more traffic processing devices (TPD) attached to a legacy router. More specifically, a TCU is defined by a router interface and the TPDs that traffic of the interface can be redirected to. Figure 9.5 shows a simple TCU consisting of one TPD connected to a router. The devices can be physically separate, even located at different sites, or integrated into future routers.

Traffic of specific network users can be redirected permanently to the traffic processing device. The traffic is processed according to the service
requested by the network user. Services are composed of components that are arranged as directed graphs [25]. Each component performs some well defined packet processing. The functionality of the components is restricted as described in Section 9.4.5 to prevent misuse of the system. A network user may define two different stages of packet processing. As discussed in Section 9.4, these processing stages determine the processing of packets having the network user’s source and destination IP address, respectively.

9.5.3 Traffic Control Unit Placement

Figure 9.6 illustrates the mitigation impact of the Traffic Control Unit (TCU) placement in a DoS attack scenario. The figure shows a graph of nine interconnected autonomous systems (AS) that form an Internet-like network. The arrows indicate the routes taken to the victim AS assuming single path routing.

The TCUs are installed at all border links of ASes marked with a rectangle in the figure and perform two types of filtering:

- **Ingress filtering**: IP packets leaving a stub AS or a multi-homed AS without transit traffic are discarded if they do not contain a valid source IP address belonging to that AS.
- **Semi-maximal route-based filtering**: IP packets with an invalid source IP address according to Border Gateway Protocol (BGP) [106] routing information (i.e. no valid route exists from the source address observed in an IP packet through the TCU’s controlled link) are discarded. It is semi-maximal as the destination address of a packet is not used in addition to the source address in order to check a route validity of a packet. We made the restriction to semi-maximal for performance reasons; for maximal filtering and \( n \) being the number of all ASes, each TCU would need to match IP addresses against \( O(n^2) \) many stored valid routes, which poorly scales. With semi-maximal filtering, only \( O(n) \) many valid routes need to be checked against and stored.

For the mitigation of the DoS attack\(^6\) depicted in Figure 9.6 launched by a computer in AS A towards a victim server in AS V, we restrict

\(^6\)A reflector (D)DoS attack can be mitigated effectively by discarding all packets
Figure 9.6: Traffic control unit (TCU) placement for DoS attack mitigation

the address range for possible spoofed source IP addresses by ingress and semi-maximal route-based filtering. Assuming many spoofed source addresses taken from ASes other than AS A, this greatly reduces the attack’s intensity. A further mitigation effect could then be achieved by rate-limiting traffic originating from the attacker’s AS respectively ASes, where we conjecture the attacker’s location. However, we do not discuss such complementary countermeasures here.

For an evaluation of the filter effectiveness, we define $S_{A,(X,Y,Z), V}$ as the set of all ASes of which IP addresses can still be used from for a DoS attack with source spoofing by an attacker in AS A to attack a

with a spoofed source address of the victim, which are not on a valid route originating from the victim. This simpler case is not further discussed for placement and performance analyses.
9.5 Infrastructure

victim in AS V if we filter at the border links of the indicated ASes X, Y, and Z. The smaller this set is, the more effective is our filtering and consequently attack mitigation.

- Without any filtering, we have
\[ S_{A,,\{\},V} = \{A, B, C, \ldots, H\} \cup S_{A,,\{\},V} = 8. \]
- When filtering at AS F as shown in Figure 9.6(a), we get
\[ S_{A,,\{F\},V} = \{A, B, C, D, E\} \cup S_{A,,\{F\},V} = 5. \]
- When filtering in AS E and F as shown in Figure 9.6(b), we can reduce it to
\[ S_{A,,\{E,F\},V} = \{A, C\} \cup S_{A,,\{E,F\},V} = 2. \]
- When filtering in AS C and F as shown in Figure 9.6(c), we get an optimum solution of
\[ S_{A,,\{C,F\},V} = \{A\} \cup S_{A,,\{C,F\},V} = 1. \]

The closer to the source we can filter, the more effective filtering becomes. Furthermore, the less collateral damage (wasted bandwidth, high packet loss, etc.) in ASes transporting attack packets accrues.

In the following, we discuss five different strategies for TCU placement:

- **All ASes:** If all ASes would install TCUs at their border links, we could completely prevent source spoofing on the AS level by enforcing worldwide ingress filtering. However, this full coverage is unlikely to happen within reasonable time due to economic, administrative and organisational restraints. Past experiences have shown that many of the worldwide distributed ASes hesitate to or only slowly incorporate new Internet technologies (e.g. IPv6, IP multicast) into their networks.

- **AS of victim:** For large-scale DDoS attacks, filtering exclusively at the AS of the victim is rather ineffective, especially if that AS is small. Neighbouring ASes will suffer collateral damage that might even prevent legitimate packets to reach the AS with the service under attack. This strategy would mean to install TCUs at least at each AS, where likely attack targets are connected to the Internet. This would mean to deploy it at least at all ASes that connect larger corporate and governmental customers.
• **Minimal vertex cover of AS graph:** A minimal vertex cover of a Graph \( G=(V,E) \) is a smallest set of vertices \( T \subseteq V \) if every edge in \( E \) is incident to some node in \( T \). Such a placement assures that we can usually filter near or even at an attacking AS. We will analyse this approach regarding the number of nodes needed in Chapter 9.5.4 and find it to be reasonably low for the Internet.

• **Randomly selected ASes:** Unlike the minimal vertex cover strategy, randomly picking ASes would completely ignore the power-law structure of the Internet [53] and pick poorly connected ASes with the same probability as other ASes. Poorly connected ASes transport less traffic and interconnect fewer routes. Consequently, considerably more ASes would have to deploy TCUs for getting a similar protection as with a minimal vertex cover.

• **Most connected ASes:** Picking the \( x \) most connected ASes is somewhat similar to the minimum vertex cover approach. However, regions of the Internet, where mostly small ASes are connected, would not get protection by TCUs. While security between heavily connected ASes would be assured, the rest of the Internet would only marginally profit from this strategy.

We conclude that the minimum vertex cover approach is clearly the most promising for TCU placement. It assures with a reasonably low number of ASes having TCUs deployed that we can usually filter close to the attack source and that we can intercept any attack route.

### 9.5.4 Traffic Control Unit Filter Efficiency

For the selection of ASes for TCU placement, we need to calculate a minimal vertex cover for the Internet AS graph. On a general graph, this problem is NP-complete [61]. However, good approximation algorithms exist:

• A greedy algorithm using the heuristic to pick the node which covers the most remaining edges not yet covered.

• An approximation algorithm described in [100] using randomisation that guarantees to find a vertex cover which is at most twice as large as a minimal vertex cover.
Thanks to the power-law structure of Internet connectivity [53], we can expect the size of a minimal vertex cover for the Internet AS graph to be reasonably small. The approximation of the minimal vertex cover for the Internet AS graph of 1997 [93] calculated as the minimum of one run of the greedy and ten runs of the randomised algorithm results in a set comprising 18.9% [101] of all Internet ASes.

Park and Lee have evaluated in [101] the efficiency of route-based filtering for the Internet AS graphs of 1997, 1998 and 1999 that can be obtained from the National Laboratory for Applied Network Research (NLANR) [93]. In spite of the rapid growth of the Internet from 3015 ASes in 1997, to 3878 ASes in 1998, to 4872 ASes in 1999 (as measured by NLANR) the power-law structure and also further structure properties based on Laplacian spectrum analyses observed from 1997 to 2001 remained stable as stated in [63]. This indicates that the relative result values (percentages) of the analyses in [101] are very likely to be still accurate for today's Internet.

We briefly summarise the findings of [101] regarding the efficiency of route-based filtering in the 1997 Internet AS graph from NLANR that are accurate for the case of our traffic control system with TCUs applying filtering as described in 9.5.3 and that are placed at the border links of all ASes being part of a minimal vertex cover.

- The IP addresses of only 12% of all ASes can be used to launch (D)DoS attacks. Addresses taken from 88% of all ASes will be detected and such packets discarded by filtering TCUs on the attack path.
- Only 4% of all (AS source, AS destination)-pairs allow for a successful (D)DoS attack using spoofed source (i.e. non-"source AS") addresses as traffic of 96% of all such AS pairs will be discarded by a filtering TCU on the attack path.
- For a filtering TCU coverage above 20% of all ASes, the fraction of forged (source AS, destination AS, source AS that "lends" spoofed addresses)-triplets divided by \( (n^2 - n) \), with \( n \) = number of ASes, reaches almost zero.
- Every DoS attack (using spoofed source addresses) can be localised to within 5 candidate AS sites. Hence, partial traceback comes almost for free when using route-based filtering.
This analyses confirm that active security components within the network can considerably increase Internet security and effectively mitigate DoS attacks even if deployed on merely about 20% of all ASes.

9.5.5 Scalability Issues

The scaling factors that our distributed traffic control service depends on are 1) the number of service subscribers (i.e. network users), 2) the total number of ISPs deploying our service, 3) the number of rules installed per network user, and 4) the bandwidth of network links. This section discusses the parameters that are influenced by these scaling factors.

Service logic and state per TPD

Each user of our traffic control service will use some specific customisable service modules (e.g. ingress filtering, traceback support) that will result in service logic and per-user state being activated in our traffic processing devices.

The traffic control service will be a charged premium service because it requires additional infrastructure to be installed and operated. We target our distributed traffic control service at large organisations that are strongly dependent on Internet communication for their revenue, for vital information exchange or their reputation: Large online shops, large companies, organisations that make heavy use of VPNs to connect their subsidiaries, business to business portals, governmental organisations and others. We do not target our service at home users or small enterprises.

In July 2004, there were 285 mill. hosts [96] connected to the Internet, roughly 26 mill. thereof were active web servers (February 2005) [95] that hosted 59 mill. web sites. For active SMTP, VPN and other common servers no reliable numbers seem to be available. We estimate the number of potential traffic control service subscribers that meet the discussed criteria to about 1% of the active web servers, which would be about 260,000. Each subscriber is expected to request a few customised services to be executed on his behalf, which results in service logic to be deployed on a number of TPDs and in per-subscriber configuration information and potentially service state to be kept on the TPDs.
Although several subscribers may request the same service to be executed on a TPD, the service logic needs to be deployed only once because the logic can be shared. However, configuration and state information, if applicable, must be kept on a per-user basis. The number of TPDs involved in a service of a particular user is service-specific. That is, some services (e.g., service level agreement validation) involve only a few TPDs, whereas others (e.g., ingress filtering) improve in quality the more TPDs are included.

If we assume that each network user registered with our traffic control service subscribes to 10 services, that the services of a user run on 1% of the TPDs on average and that each service includes per-user state, we find that each TPD must keep the configuration information and state for 2,600 users. Accounting for 1 kByte of configuration and state information per user and service, this results in 26 MByte of memory needed per TPD, which is a rather modest requirement.

**Signalling Effort**

The number of ISPs offering distributed traffic control influences the number of service deployment messages that need to be sent by the TCSP to the ISP management stations each time a network user requests the provisioning of a new service. According to the CIDR report [14] of August 2005 there were 20,107 autonomous systems, out of which 8,323 announced only one network prefix. Even if we assume that each AS corresponds to one ISP and that all ISPs offer our distributed traffic control service, signalling overhead due to the secure distribution of the small service deployment messages by the TCSP to a few thousand ISPs is not a bottleneck. Furthermore, proactive services like ingress filtering do usually not require instant deployment.

**Traffic Processing Capacity**

Due to high performance demands, a hardware-based solution for our traffic processing devices is favourable. Research prototypes of FPGA based devices exist that can concurrently filter 8 mill. flows [112] on a 2.5 GBit/s (OC-48) link. According to [10], faster FPGAs allow achieving advanced packet filtering at 10 GBit/s (OC-192). Highly flexible solutions using commercial network processors are an alternative.
End-to-end Principle Argument

The *end-to-end principle* in system design [108] favours a network with a simple but powerful packet forwarding service. However, in [105] the authors argue this principle should be interpreted on a case by case basis. We are convinced that the increase in network security and functionality outweighs the disadvantages of the complexity added to the network by deploying our traffic control service.

9.6 Use Case

In this section, we give a sample scenario for a DDoS mitigation service, describe the deployment steps and explain the service mapping process using XML documents.

9.6.1 DDoS Mitigation Service

We assume that a (fictional) U.S. based company named Xorus operates a large online shop “XorusPerfumes.shop”. All its revenues stem from online sales and therefore Xorus is interested in having its shop available 24/7 without interruption. After some recent DDoS attacks on some other large online shops, Xorus registers for the traffic control service at the TCSP. It installs some proactive traffic control services that filter out UDP based attacks towards their web site’s IP address right at the source as there is no need for UDP packets to be received and processed by the Xorus online shop. In addition, Xorus requests a rate limiting service for TCP SYN requests, which will automatically become active for non-U.S. IP addresses after a trigger condition on the rate of TCP SYN packets yields true on certain links connecting the rest of the world to the U.S. Internet backbone. This allows to provide a high service quality for U.S. customers even while under attack from abroad.

However, early on a Monday, the Xorus shop is heavily flooded by TCP RST packets from a DDoS reflector attack that misuses tens of thousands of compromised computers as DDoS agents and U.S. based web servers as reflectors. This situation is illustrated in a slightly simplified setting in Figure 9.7. Network congestion and packet loss occur not only at the victim’s site but also close to reflectors and at routers that use low bandwidth peering links as indicated with the flash symbol. A possible
reaction would be to activate inbound rate-limiting or manually blocking
known source IP addresses at the victim’s downstream link. However,
this will completely deny access of the victim to services offered by the
reflector agents, and even worse, collateral problems within the Internet are
not addressable with such a strategy. For short, the victim’s service
under attack might still appear to be offline for legitimate users.

Instantly, Xorus as a customer of the traffic control service reacts and
deploys a worldwide ingress filtering service on its shop’s IP addresses
as illustrated in Figure 9.8. This action blocks all IP packets sent over
an Internet uplink with the shop’s IP addresses from a location different
than Xorus’ own uplink. The attack is immediately stopped. In addition,
Xorus deploys a logging service at the uplinks to trace attack traffic back
to the real origin of the reflector attack in the hope to find the attacking

Figure 9.7: A DDoS reflector attack hits Xorus’ online shop.

Figure 9.8: DDoS reflector attack mitigation at the source using the
traffic control system
agents and possibly also the attacker. The Internet traffic policemen in Figure 9.8 stand for the new power a victim has within the Internet and close to the attacking sources thanks to the traffic control system.

9.6.2 Service Deployment

For our further explanations, we focus on the ingress and route-based filtering service for the scenario of Section 9.6.1, which is deployed to mitigate the attack based on reflected TCP RST messages. As our network user (Xorus) has already registered with the TCSP, it can directly initiate the deployment of the ingress filtering service. Therefore, Xorus selects the service from the TCSP’s web site together with service-specific parameters. Client authentication is used to make sure that only a legitimate network user can initiate service deployment. A service request is generated and used as input for the service mapping process in the TCSP layer. Alternatively, such a service could be deployed beforehand and activated later automatically upon fulfilment of a trigger condition (e.g. detection of a DDoS flooding attack).

Figure 9.9 shows the details of the deployment process, which is subdivided into TCSP, ISP, TCU and Device layer. The complete deployment process is carried out at the management stations of TCSP and ISP. For each service that a layer offers, a service descriptor specifies the following:

- The mapping of the service to sub-services offered by the layer below.
- The set of mandatory and optional parameters, their default values and their mapping to parameters for sub-services.
- Restrictions that direct the placement of service logic based on context information available at different layers.

For each layer a database contains context information about the infrastructure relevant to that layer\(^7\). Information at the TCSP layer includes the identities of contracted ISPs and properties of their networks, e.g. whether they transport transit traffic or provide a stub network. At

\(^7\) These logical databases may be merged into two physical databases located at the TCSP and ISP management station.
the ISP layer relevant information includes the location of the TCUs, e.g. whether a TCU is located at the border of a network or in the core network. At the TCU layer details about the pairing of TPDs and routers as well as BGP routing information (e.g. for route-based filtering) are kept as context information. Finally, at the Device layer information about the make and version of TPDs and routers and their configuration interfaces must be kept. Additionally context information can contain dynamic state information about managed objects and deployed services.

Deployment logic on each layer maps the service request from the layer above to services provided by the layer below (right column of Figure 9.9) based on information provided by the service descriptors (left column of Figure 9.9) and generates corresponding lower layer service requests. Taking into account restrictions specified in the service descriptor and context information from the databases, sub-services are placed on the managed objects of the corresponding lower layer (ISPs, TCUs, TPDs and routers, respectively). The middle column of Figure 9.9 describes the mapping process at the different layers as carried out for the ingress filtering service. The deployment process ends with the configuration of the devices that were previously selected to run part of the service logic. Composition and deployment issues of distributed network services are investigated and discussed in detail in the PhD thesis of Matthias Bossardt [23].
154 CHAPTER 9. TRAFFIC CONTROL SERVICE

Figure 9.9: Service mapping

Service request from network user

- **Service Name**: "ingressFilter"
- **Target Type**: "source"
- **Address**: 82.130.103.1/24
- **Traffic Type**: "IP"
- **Parameter**: "trafficType" = "IP"
- **Use Value**: 130.59.36.251

TCSP Layer

- The network user requests an ingress filter service to be applied to packets with source address in the range 82.130.103.1/24.
- The service descriptor specifies that the service block traffic to addresses of the specific router interface (uplinkRouterIf). This useValue is passed on in the sub-service request.
- The service is implemented using an uplinkRouterIf sub-service.
- The restriction with scope "ISP" constrains the placement of the service to ISPs that provide stub networks. The type of networks offered by ISPs is retrieved from the context database.
- The other restrictions concern the TCUs and therefore passed on.

ISP Layer

- At this level, we are only dealing with ISPs offering stub networks due to the selection at the TCSP layer.
- As specified in the service descriptor, the requested ingress filter service is implemented using a tpdBlock sub-service.
- The notification with scope TCU redirects the sub-service request that the tpdBlock service is only installed on TCUs whose linkType is "UP". An access point link connects the network of a customer to the ISP's network. The linkType is retrieved from the context database.
- The second restriction of the request avoids that the ingress filter service is configured for the uplink interface (uplinkRouterIf) of the service requester.
- The tpdBlock service is configured with "incoming" in order to block incoming traffic.

TCU Layer

- The tpdBlock service is split into two sub-services on the device layer: the routedRedirect service and the tpdBlock service.
- The destination address of the redirection is retrieved from the context database.

Device Layer

- A routedRedirect service is installed on the router.
- The service requests are translated into device-specific configuration commands. The name and model of the devices are retrieved from the context database.
9.7 Conclusions and Future Work

Our analysis of earlier proposed DDoS attack mitigation systems revealed several inherent weaknesses, which impede those systems to cope with certain classes of DDoS attacks. In particular, such systems may completely cut off legitimate servers or networks under a DDoS reflector attack, thus amplifying the effects of the attack.

We presented our patented distributed traffic control system based on the concept of traffic ownership that enables ISPs to deploy new applications within the network and to safely delegate partial network control to network users [43]. We described how such a system can be used to prevent DDoS reflector attacks, which earlier proposed DDoS attack mitigation systems failed to counteract as our analysis showed. Ultimately, our system effectively stops attack traffic close to the source. Herewith, it frees network resources that are nowadays wasted for transporting attack traffic around the globe and that harm not only the target system but also cause collateral damage like network congestion. Many new applications, also not security related ones, are expected to emerge once such a system is available.

Leveraging acceptance by network operators for such a system will be vital. We think that our traffic control system offers many incentives for network operators and at the same time a high level of security against misuse, which was a major concern with other approaches in the field of active and programmable networks.

We are convinced that incorporating more security within the Internet core is feasible and reasonable. In medium term, we expect that network users demand a more secure Internet from the network operators and such a traffic control system would be one possible answer.
Chapter 10

Conclusions and Outlook

10.1 Conclusions

We have addressed four core problems:\footnote{The full problem statements are given in the Introduction.}

1. Companies and governments are not aware of the possibly huge economic impact of large-scale Internet attacks.

2. Only little is publicly known about the characteristics of real past Internet worms.

3. Due to a lack of comprehensive Internet backbone online monitoring systems that can early detect large-scale attacks, reaction to new attacks by network operators is slow, cumbersome and possibly counterproductive.

4. Effective countermeasures against large-scale distributed Internet attacks like DDoS\footnote{DDoS stands for Distributed Denial of Service.} currently do not exist. Attack defence is ex-
executed mostly close to the victim instead of close to the attack sources.

We give our main findings and conclusions, which are all related to research on large-scale Internet attacks, for each of the above problem statements:

1. We presented our economic damage model for the scenario of a massive DDoS attack, which results in an interruption of Internet services. Using this model, we estimated the economic damage for specific Internet dependent companies as well as for the whole Swiss economy. We found that economic damage usually has not the same characteristics over time as technical problems have. Economic damage can still grow when technical problems have been resolved and the attack has stopped. Damage estimates by our model can be used for assessing the vulnerability of Internet dependent companies, for optimising investments into safeguards and preventive security measures, and to assess a “cyber risk” insurance policy. In addition, we could show that spam causes significant economic damage as well. We came to the conclusion that the Internet is about to become a critical infrastructure due to the high dependence of modern economies. In order to protect the Internet from such attacks, first the awareness of its users and operators for the large impact of a massive attack must be further raised.

2. We analysed two major past Internet worm outbreaks, namely the network worm Blaster.A and the e-mail worm Sobig.F. We used flow-level traffic data exported from all border routers of AS559 (SWITCH) that were captured in the DDoSVax [134] project. Our network-centric worm analyses help to better understand the impact of worms on the Internet and give a basis for elaborating new worm outbreak detection methods. We found many interesting details about the spreading events, deployment of countermeasures by network operators and timing characteristics of scanning activities and successful infections. We could show that catching actual worm code of a multi stage and multi protocol worm like Blaster in the backbone is almost impossible even during the initial outbreak. We found that there were considerable time-skews of one to several hours from the visibility of the worm spreading activities by many
external infected computers in the backbone traffic until a first successful infection happened over the observed backbone. Our findings support the importance of early detection mechanisms.

3. We developed a framework (“UPFrame”) for monitoring backbone traffic. We demonstrated its applicability with several plugins that analyse network state and find anomalies. Our generic host behaviour based method showed a strong sensitivity for worm outbreak events and was able to early and reliably detect past major network and e-mail based Internet worms.

4. As a first step towards more security within the network, we proposed the fundamental concept of traffic ownership. It greatly alleviates the danger of losing control over the network and eliminates privacy concerns caused by the deployment of active programmable components in the network. We designed a worldwide traffic control system, which can safely delegate partial network control from network operators to network users. In a use case, we illustrated how it could be used against DDoS reflector attacks by stopping the attack at its sources. Our traffic control system provides a versatile platform that could be used also for many other network based services such as, e.g., inter-domain service level agreement validation.

From our research findings we can derive that:

- Potential costs arising due to a massive attack that disrupts Internet services can be huge as shown by our economic damage model scenarios. Using our model, transparent calculations to assess such potential damage could now be given by any Internet dependent enterprise.

- By uncovering the huge potential cost of an attack for an enterprise or entire nation, we could show a clear need from the network user’s perspective for improved Internet security that helps to prevent large-scale attacks.

- Having an early warning system in place could help to considerably shorten response time and consequently to reduce damage suffered.
An early attack warning system based on monitoring flow-level traffic crossing border routers of a backbone is feasible. As such flow-level data is already in widespread use for accounting purposes at most larger backbone providers, it is readily accessible at the provider’s premises.

The largest obstacles for providers to become more active in early attack detection seems to be that they get reimbursed for each byte transferred. Consequently massive attacks, the majority of which are not massive enough to be critical for backbone operation, are currently favourable to a provider’s revenue and motivation to stop attacks is therefore low. Changing this reimbursement scheme could be achieved by enforcing service level agreements assuring a high quality of Internet service with contractual penalties in the case of quality deviations or if Internet users would emphatically request a new premium attack-free Internet uplink service.

Security within the network as explicated by our Traffic Control System is much more powerful than at the network border. Filtering closer to the attack sources is much more effective and prevents also collateral damage compared to filtering only at the attack victim as usually done today. This requires new security technology within the network.

Our traffic control system could be incrementally deployed and used to help prevent and fight attacks within the network core and close to the attack sources. Using our fundamental concept of traffic ownership, a network having such a system deployed remains manageable by the network operators although that network users get powerful control also within the Internet.

In the near future, we expect a major shift of the responsibility for enforcing Internet security from network users to network operators as Internet users will no longer accept to be attacked almost daily.

In summary, we found that incorporating more security within the Internet core in order to early detect and mitigate large-scale Internet attacks is feasible, effective and reasonable.
10.2 Outlook

Further research to improve Internet security is required to deal with new and raising threats caused by the huge and still growing popularity of the Internet. We highlight some possible research topics for possible future work:

- Analyse further major and minor worm outbreaks and attacks using the DDoSVax NetFlow archive.
- Design further algorithms for the early detection of massive and large-scale Internet attacks based on backbone traffic observation in order to reduce attack response time.
- Find approaches to (semi-)automatically set reasonable static and dynamic warning thresholds for attack detection algorithms in various different network environments.
- Improve e-mail and VoIP based communication to prevent spam and spit\(^3\).
- Investigate the effects of the Internet Protocol version 6 (IPv6) on network security incident detection and countermeasures. Consider its large 128 bit host address space and the fact, that a full implementation of IPv6 requires a client to support the Encapsulating Security Payload (ESP) header defined by IPSec, which consequently allows the transmission of e.g. malcode in encrypted form directly from end-host to end-host. End-to-end encryption of malcode transmissions will render most packet inspection based intrusion detection systems useless.
- Develop new security mechanisms and concepts that make worldwide information interchange safe, reliable and free of malicious code and attacks.

\(^3\text{Spit stands for spam over Internet telephony}\)
Acknowledgements

I thank Professor Dr. Bernhard Plattner for enabling and supporting the research presented in this thesis and for offering a relaxed and enjoyable work environment to his PhD students. I thank Arno Wagner (ETH) for his efforts to build the DDoSVax NetFlow archive, designing and operating the Scylla cluster, creating libraries and tools for offline NetFlow data processing and many fruitful discussions. I thank Matthias Bossardt and Elisa Boschi for the inspiring collaboration on the traffic control system. I thank members of the Communication Systems Group at ETH for feedback and interesting discussions, especially (in alphabetical order) Rainer Baumann, Daniela Brauckhoff, Károly Farkas, Uli Fiedler, Placi Flury, Stefan Frei, Jan Gerke, Hasan, David Hausheer, Kostas Katriinis, Vincent Lenders, Martin May, Georgios Parissidis, Marc Remnhard, Lukas Ruf, Daniel Sigg and Bernhard Tellenbach. I thank the Computer Engineering and Networks Laboratory (TIK) secretaries and the TIK services group for providing friction-less administrative and excellent technical services, and Lukas Ruf for providing the \LaTeX{} thesis template. I thank researchers at ZISC, the Zurich Information Security Center, especially Germano Caronni and Michael Näf, for interesting research discussions.

I thank the students that I supervised or co-supervised during their student thesis for their contributions to the DDoSVax and other security related projects, namely (in alphabetical order and annotated with the title of their student thesis): Jonas Bolliger (Detecting Bots in Internet Relay Chat Systems), René Gallati (Near Real-Time Detection of Traffic Usage Rhythm Anomalies in the Backbone), Christoph Göldi (Scan Detection Based Identification of Worm-Infected Hosts), Emil Haraldsson (DDoS Attack Detection Based on NetFlow Logs), Lukas Haemmerle (P2P File-
sharing Population Tracking), Markus H"acki (Kostenanalyse von pr"aventi-veen IT-Sicherheitsmassnahmen), Bodo Hechelmann (Kostenmodell zu pr"aventi-ven IT-Sicherheitsmassnahmen), Roman Hiestand (Simulation-Based Analysis of Internet Worm Characteristics; Scan Detection Based Identification of Worm-Infected Hosts), Theus Hossmann (Analysis of Sobig.F and Blaster Worm Characteristics), Franco Hug (Implementation of a Distributed Traffic Control Service using FPGAs), Philipp Jardas (P2P Filesharing Systems: Real World NetFlow Traffic Characterization), Christoph Jossi (Management of Distributed Traffic Control Service), Thomas Kaufmann (Detecting Bots in Internet Relay Chat Systems), Roger Kaspar (P2P File-Sharing Traffic Identification Method Validation and Verification), Matth"ias Keller (Web Attack Showcases for the Security Demo Lab), Roger Lacher (Secure Java Code Interpreter for Public Utility Computing), St"ephane Racine (Analysis of Internet Relay Chat Usage by DDoS Zombies), Daniel Reichle (Analysis and Detection of DDoS Attacks in the Internet Backbone using NetFlow Logs), Caspar Schlegel (Realtime UDP NetFlow Processing Framework), J"urg Schmid (Wirtschaftlicher Schaden von DDoS Attacken auf Backbone-Provider), Bernhard Tellenbach (Visualisation of Client/Server Behaviour), Peter Weigel (Wirtschaftlicher Schaden von DDoS Attacken auf Backbone-Provider), Rashid Waraich (Attack Signature Generation Survey), Andrea Weisskopf (Plug-ins for DDoS Attack Detection in Realtime), and Gerry Zaugg (A Lightweight Packet Capturer for High-Speed Links).

I am grateful to ETH, SNF (grant 200021-102026/1) and SWITCH for financing the DDoSVax project. I thank our industry partners Peter Haag, Simon Leinen, Christoph Graf and Willi Huber from SWITCH for continuously providing us with NetFlow data and support and Stefan Lampart, Roel Vandewall and Martin Bosshardt from Open Systems for jointly supervising students with us. I thank the anonymous reviewers for their helpful feedback.

Last but not least, many thanks go to my parents, brothers and sisters, and my girlfriend for their unconditional generosity and all others that influenced the research of this PhD thesis and that were not explicitly mentioned here.
Curriculum Vitae

Thomas Dübendorfer
Dipl. Informatik-Ing. ETH
thomas@duebendorfer.ch
http://thomas.duebendorfer.ch/

Born May 1st, 1975
citizen of Bassersdorf (ZH), Switzerland

Education
2001 – 2005 ETH Zürich, PhD studies in network security
2003 ISC² CISSP security professional certification
1999 – 2001 ETH Teaching Certification in Computer Science:
Didaktischer Ausweis ETH (höheres Lehramt)
1995 – 2001 ETH Zürich, Diploma (M.S.) in Computer Science
1988 – 1994 Matura type B, Grammar School Zürich Oerlikon

Work Experience
2000 – present ETH Zürich, research and teaching assistant
2002 – present Member of the board of directors, Dübendorfer AG
4. – 8.2001 Hewlett-Packard Research Labs, Palo Alto, USA, internship
1998 – present Security consulting projects for major Swiss banks,
government and courts
1998 – 1999 Netcetera AG, Switzerland, software engineer
Payserv AG, Switzerland, software engineer
Honours

2005 SWITCHaward 2005 nomination for project DEMIAN\(^a\) out of 42 applicants
2005 Best paper award from IEEE at WET ICE/STCA 2005 for "Host Behaviour Based Early Detection of Worm Outbreaks in Internet Backbones"
2004 Best paper award from IEEE at WET ICE/ES 2004 for "An Economic Damage Model for Large-Scale Internet Attacks"
2001 Willi Studer prize for best M.S. ETH in Computer Science 2001 and ETH medal for Master's thesis
1997 Top Five, 2nd pre-diploma
1996 Top Five, 1st pre-diploma
1994 Best matura type B (of 65 candidates) of the year, grammar school Zurich Oerlikon

\(^a\)Project DEMIAN is joint research with M. Bossardt and is about a distributed Internet traffic control service based on the new concept of traffic ownership (see also part III - “Mitigation” in my PhD thesis).
Appendix A

NetFlow v5 Format

The following two Tables A.1 and A.2 give the header and record fields of the Cisco NetFlow v5 [36] flow format.
<table>
<thead>
<tr>
<th>Bytes</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>version</td>
<td>NetFlow export format version number</td>
</tr>
<tr>
<td>2-3</td>
<td>count</td>
<td>Number of flows exported in this UDP packet (1-30)</td>
</tr>
<tr>
<td>4-7</td>
<td>SysUptime</td>
<td>Current time in milliseconds since the export device booted</td>
</tr>
<tr>
<td>8-11</td>
<td>unix_secs</td>
<td>Current count of seconds since 0000 UTC 1970</td>
</tr>
<tr>
<td>12-15</td>
<td>unix_nsecs</td>
<td>Residual nanoseconds since 0000 UTC 1970</td>
</tr>
<tr>
<td>16-19</td>
<td>flow_sequence</td>
<td>Sequence counter of total flows seen</td>
</tr>
<tr>
<td>20</td>
<td>engine_type</td>
<td>Flow-switching engine type (RP,VIP, etc.)</td>
</tr>
<tr>
<td>21</td>
<td>engine_id</td>
<td>Slot number of flow-switching engine</td>
</tr>
<tr>
<td>22-23</td>
<td>sampling_interval</td>
<td>First two bits hold the sampling mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00 = no sampling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01 = sample one of every ( x ) packets and place it in the NetFlow cache</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 and 11 reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Last 14 bits hold sampling interval</td>
</tr>
</tbody>
</table>

**Table A.1:** NetFlow version 5 header format
<table>
<thead>
<tr>
<th>Bytes</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>srcaddr</td>
<td>Source IPv4 address</td>
</tr>
<tr>
<td>4-7</td>
<td>dstaddr</td>
<td>Destination IPv4 address</td>
</tr>
<tr>
<td>8-11</td>
<td>nexthop</td>
<td>IPv4 address of the next hop router</td>
</tr>
<tr>
<td>12-13</td>
<td>input</td>
<td>SNMP index of the input interface</td>
</tr>
<tr>
<td>14-15</td>
<td>output</td>
<td>SNMP index of the output interface</td>
</tr>
<tr>
<td>16-19</td>
<td>dPkts</td>
<td>Number of packets in the flow</td>
</tr>
<tr>
<td>20-23</td>
<td>dOctects</td>
<td>Total number of IP (layer 3) bytes of the flow’s packets</td>
</tr>
<tr>
<td>24-27</td>
<td>First</td>
<td>SysUptime at the start of the flow (in milliseconds)</td>
</tr>
<tr>
<td>28-31</td>
<td>Last</td>
<td>SysUptime at the end of the flow (in milliseconds)</td>
</tr>
<tr>
<td>32-33</td>
<td>srcport</td>
<td>TCP/UDP source port number or equivalent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For ICMP: srcport is set to 0</td>
</tr>
<tr>
<td>34-35</td>
<td>dstport</td>
<td>TCP/UDP destination port number or equivalent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For ICMP: dstport = (ICMP type) \cdot 256 + (ICMP code)</td>
</tr>
<tr>
<td>36</td>
<td>pad1</td>
<td>Unused (zero) byte</td>
</tr>
<tr>
<td>37</td>
<td>tcpflags</td>
<td>Cumulative OR of TCP flags$^a$</td>
</tr>
<tr>
<td>38</td>
<td>prot</td>
<td>IP upper layer protocol (for example, 6 = TCP, 17 = UDP)</td>
</tr>
<tr>
<td>39</td>
<td>tos</td>
<td>IP type of service (ToS)</td>
</tr>
<tr>
<td>40-41</td>
<td>src_as</td>
<td>Autonomous system number (AS) of the source IP address</td>
</tr>
<tr>
<td>42-43</td>
<td>dst_as</td>
<td>Autonomous system number (AS) of the destination IP address</td>
</tr>
<tr>
<td>44</td>
<td>src_mask</td>
<td>Source address prefix mask bits</td>
</tr>
<tr>
<td>45</td>
<td>dst_mask</td>
<td>Destination address prefix mask bits</td>
</tr>
<tr>
<td>46-47</td>
<td>pad2</td>
<td>Unused (zero) byte</td>
</tr>
</tbody>
</table>

$^a$Set to zero in DDoSvax archived flows due to router restrictions

**Table A.2:** NetFlow version 5 record format
Bibliography


