A Detection and Filter System for Use Against Large-scale DDoS Attacks in the Internet Backbone

Lukas Ruf, Arno Wagner, Károly Farkas, Bernhard Plattner

Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology (ETH) Zurich
CH-8092 Zurich/Switzerland
{ruf,wagner,farkas,plattner}@tik.ee.ethz.ch *

Abstract. Distributed denial of service (DDoS) attacks in the Internet pose huge problems on nowadays communication infrastructure. Attacks either destroy information or impede access to a service. Since the significance of the Internet to business and economy is growing rapidly, efficient protection mechanisms are urgently required to protect hosts from being infected and, more important, sites from being attacked. Detection of DDoS attacks requires deep packet inspection at link speed, and context-dependent packet handling for countermeasures. This functionality is not achievable with nowadays commercial high-performance routers.
In this paper, we therefore present our problem space exploration of DDoS attacks and propose a flexible service architecture for detection and filter mechanisms to counteract DDoS attacks. To achieve the performance required for backbone routers together with the flexibility needed for services counteracting DDoS attacks, we base the proposal on our PromethOS NP router platform that manages and controls hierarchical network nodes built of network and host processors.

1 Introduction and Motivation

Present day communication infrastructure has been seriously threatened by large-scale distributed denial of service (DDoS) attacks in the Internet. These attacks destroy information or hinder customers from accessing specific services. Services provided in the Internet like on-line stock trading, virtual travel agencies or book-stores are very important to economy already today. The Economist

* This work is partially sponsored by the Swiss Federal Institute of Technology (ETH) Zurich, the Swiss Federal Office for Education and Science (BBW Grant 99.0533), the Swiss National Science Foundation under Grant 200021-102026/1 and Swiss Academic Research Network (SWITCH). PromethOS v1 has been developed by ETH as a partner in IST Project FAIN (IST-1999-10561). Moreover, we would like to acknowledge the great support received from the IBM Zurich Research Laboratory, and we acknowledge the valuable feedback and comments by the reviewers.

reported in May 2004 [31]: "The 200m Americans who now have web access are likely to spend more than US$120 billion online this year." But in eCommerce, brief inaccessibility of services results in loss of business [31]. Since the impact of eCommerce on economy is expected to grow further, the risk of economic damage resulting from a large-scale Internet attack increases [11]. The situation becomes more dramatic because the number of attacks increases at least at the same pace as the impact of eCommerce does. Of further threatening importance is the fact that newly discovered errors in soft- or hardware are exploited more rapidly for fresh attacks [32].

The effect of large-scale DDoS attacks in the Internet correlates with the number of infected hosts that launch attacks towards other sites. Hence the threat emerges as more and more private and insufficiently managed hosts are connected to the Internet by broadband lines. Home users are rarely aware of the problems and dangers in the Internet nor are they able to manage and protect their hosts effectively. But eCommerce flourishes not at least thanks to the widespread use of the Internet by home users [31]. Companies afford security and system administration teams quite often, but they suffer from similar problems.

To effectively protect the Internet, hosts need to be protected from becoming an attacker as well as from being attacked at any site. It is hard if not unfeasible to install protection mechanisms at this level of granularity without blocking daily business. Hence, detection and countermeasures are required that approach this problem at the level of border routers or gateways to protect larger areas in the Internet.

Fighting DDoS attacks requires in-depth packet inspection to identify malicious streams in the flood of traffic. With today's commercial high-performance routers, however, payload analysis is not possible, usually. Or if it is, the functionality is coded either in firmware or hard-wired in the box. Attack schemes vary a lot over time. In addition, the period becomes shorter between the first detection of an exploit and the widespread launch of the attack. So, it is crucial that large-scale DDoS attacks are defeated on routers as close to the core of the Internet as possible. Specific Anti-DDoS components must be installed, configured and removed on request. For obvious reasons, the deployment of the specific detection and countermeasure components must not interfere with other services. Further, they must be able to tackle the problem of known as well as unknown attacks semi-automatically according to predefined policies.

Active Networking (AN) [30] has proposed the concept of execution environments (EEs) to address the challenges of exchanging and extending service functionality on the routers at run-time. So far, EEs have been instantiated only on a single general purpose processor (GPP) as found in legacy personal computers. But single GPP configurations are not able to cope with the demands of nowadays border or backbone traffic in the Internet. To increase the degree of programmability and simultaneously of flexibility at interface level, processor manufacturers have proposed the architecture of Network Processors (NPs) [14,15,22] to be embedded in network interface cards (so-called NP-blades). Built of con-

trol and packet processors[1], they provide additional processing capabilities and capacity in addition to the host processors. A hierarchical network node provides, thus, a perfect hardware platform for the envisioned Anti-DDoS service since packet processors are able to process packets at line rate, and processors on upper tiers provide the management and control functionality besides room for further packet processing. However, it is extremely difficult to provide a dynamically extensible router platform that provides the required abstractions and is able to manage a hierarchical network node if component based services must be able to span all tiers of the processor hierarchy. We propose PromethOS NP [24,25] as the dynamically code-extensible router platform for the envisioned Anti-DDoS service. It provides the abstractions required for node-internal communication among service components by which services are allowed to span arbitrary processors. Further, it provides the mechanisms to install, configure, instantiate and remove service components on any code-extensible processor of the processor hierarchy. Hence, the goal in this paper is to propose an architecture of an Internet backbone Anti-DDoS service for our powerful PromethOS NP router architecture.

Therefore, we structure the remainder of this paper as follows: in section 2, we present a problem-space exploration of detection mechanisms and countermeasures against large-scale DDoS attacks in the Internet to extract commonalities required for our service architecture. We briefly present the concepts and architecture of PromethOS NP in section 3. In section 4, we propose our Anti-DDoS service architecture for PromethOS NP, and present related work in section 5. Our paper is concluded by section 6, in which we give a summary and an outlook to further work.

2 Large-scale Internet Attacks

The main type of large-scale Internet attacks we focus on here is an initial worm-driven [29] compromise of a large number of hosts, followed by an optional Distributed Denial-of-Service (DDoS) attack that uses the freshly compromised hosts as attack platform. We identify three main activities [32] during this type of attack: target identification, target infection and DDoS attack. The first two activities together are also called *worm propagation*. Worm propagation can sometimes also be done in a single step, e.g. when host probing and compromise can be done with a single data packet.

The attack activity can be started by a trigger, for example a time, reception of a message from an attack control network (see e.g. [33]) or completion of a specific number of infection attempts. It can be done in parallel to worm

[1] NP vendors do not use a consistent naming scheme to refer to the code-extensible processors: the Intel IXP-architecture refers to the first-level processors as *microengines* while the IBM PowerNP identifies them as *picoprocessors* or *core language processors*. Second-level processors are named differently, as well. For this reason, we refer to the first level of processing engines as *packet processors* and to those of the second level as *control processors*.

propagation, however this usually impacts worm propagation speed negatively and is generally not done.

We now describe the basics of the attack model in more detail and identify common characteristics.

2.1 Activity 1: Target Identification

A vulnerable host offers network functionality that can be compromised. The vulnerability can be located in an application, for example a P2P filesharing client or web server, or in the operating system itself, e.g. in the network stack. It is also possible to use several different vulnerabilities in worm propagation. In order to recognize that a host is vulnerable, a vulnerable network functionality has to be found on it. This is done by sending a specific probe over the network. Probes consist of one or several specifically constructed packets that are sent to a host. A probe can consist of several sub-probes.

2.2 Activity 2: Target Infection

After a vulnerable host has been identified, it still needs to be compromised. This is done by using the vulnerability to transport to and start exploit code on the target. This may involve a multi-stage process where several steps are needed, each involving specific network activity. The end result is that the work code runs on the target host and is able to propagate further from it. Note that no complete host compromise is needed. Compromising a network application, e.g. an email client, or part of an operating system may already be enough. As an extreme case worms that use vulnerabilities in other worms (that have previously infected the target) exist. The new host is now called *infected*.

A border case is single packet propagation, were target identification and compromise are done with a single network packet, e.g. the Sapphire worm [5] needs only a single UDP packet of 404 Bytes for a successful propagation step. Single packet infection requires the use of a protocol that can transport data in the first packet, like UDP. Many vulnerabilities also do not allow single packet propagation, e.g. because several data transfers are needed. Code Red [3, 9] is an example of a worm that uses TCP with its three-way handshake [23]. As an example of a multiple protocol infection, the Blaster worm [6], uses TFTP [28] to retrieve code in a second step of the infection, after an initial exploit was used to initiate the second step.

2.3 Activity 3: Attack

The third step is to execute one or several attacks. Sets of compromised hosts have also been used for other purposes, e.g. as relay for unsolicited commercial email (SPAM), which is of interest to organized crime. This worm creation purpose has been predicted by Schechter and Smith in [26] and recently been confirmed to exist in practice by the German computer magazine c't [8]. In this

paper we only deal with the use of compromised hosts as attack platform for DDoS attacks[2].

2.4 Detection and Possible Countermeasures

We differentiate between the terms *byte-pattern*, *flow-pattern* and *traffic-pattern*. A *byte-pattern* is a sequence of bytes within a packet. A *flow-pattern* is a sequence of packets that together forms a specific attack. A *traffic-pattern* is an aggregation of multiple flow-patterns that target the same site. We do attack detection by trying to observe traffic anomalies. We argue that traffic-pattern need to be analyzed for this. For the identification of packets belonging to a specific attack, attack signatures need to be determined. Attack signatures can be detected either by byte-patterns or flow-patterns. Once this has been done for a specific attack, a countermeasure can be selected and activated. We base our detection and countermeasure service on four fundamental functional elements named *Capture*, *Identification*, *Filter* and *Slowdown*.

Data Capturing: In order to detect a worm during its propagation phase in a high-speed network, access to more than abstracted traffic data (e.g. NetFlow [7]) is desirable. One promising possibility is to obtain information about specific suspicious traffic from abstract data without payload information and then capture concrete packets to gain more insights. As an example, transferred worm code looks the same in most observed worms. If, e.g., a lot of TFTP transfers are observed, it would be desirable to find out whether most carry the same payload. Furthermore, it is desirable to capture complete instances of the transferred code. The same is true for the exploits used and for the packets sent in a DDoS attack. This information can then not only be used to better understand the worm, but is also essential in generating specific filters or slowdown mechanisms and in identification of infected hosts.

Identification of Compromised Hosts: One countermeasure desirable is the filtering of all traffic from infected hosts. This serves to block further infections as well as attacks or other misuse of compromised hosts. It also serves to force host operators to repair the compromised host software. In order to allow host filters, infected hosts have to be reliably identified. Generally, this needs payload information. The reason is that worms and DDoS attacks frequently use protocols that are also used for other purposes. Without payload based detection of infected and attacking hosts, the number of wrongly identified and blocked hosts could be large and this type of countermeasure can do more damage than good.

[2] Since worm spreading and host infection might be the attack itself, we refer to the combination of all three activities by the term *attack* if we do not state the different activities explicitly.

Filters: Besides host blocking, it is desirable to filter attack traffic out based on protocol and payload information. One reason is that the set of compromised hosts may not always be identified fully, for example if some infected hosts do not propagate the worm but wait silently after infection until they start to participate in a DDoS attack. In a filter, it may, e.g., be desirable not to block all HTTP traffic to a site under attack, but just a specific query or query type as emitted by some known infected hosts. In order for this to work, byte-patterns have to be identified and then a filter for these patterns has to be constructed and installed on the fly.

Slowdown: A variant of a filter is a slowdown filter. Instead of dropping all packets, it limits the bandwidth for packets or connections matching a signature. The advantage is that legitimate use of the target over the network is still possible, but slower. This is especially useful when attack traffic cannot be reliably identified. Filtering infection traffic to implement slowdown is hard. Fast worms as have been observed in the recent past compromise most vulnerable hosts in a matter of minutes. Still filtering infection traffic is worthwhile, since worms have a tendency to stay active for months or longer and cause both network load and new infections of the occasional newly installed and unpatched hosts. While filters have a very high disruptive potential if used incorrectly or triggered by an attacker as a type of indirect attack, slowdown is far more benign. Slowdown filters may even be safe enough to be employed in an automatic fashion, at least initially. Countermeasures will still require some human input at some time for near future. However, they can buy humans time to think and to understand what is happening.

3 PromethOS NP Router Platform

We propose the PromethOS NP [24, 25] router platform to introduce, map and accommodate services, such as our Anti-DDoS service, on a hierarchy extended active network node. Services as built of service components may span all processing elements if required. To pave the way for the Anti-DDoS service architecture, we briefly present the component based service model used on PromethOS NP and the architecture of PromethOS NP with emphasizing specific components that are required to control and manage this platform.

3.1 Component Based Service Model

Fig. 1 visualizes the service model of PromethOS NP by a configuration that illustrates the capabilities of the model. Services for PromethOS NP are described as a graph of edges and vertices. Edges represent service components (emphasized with named boxes in Fig. 1) and vertices denote interconnection points. Service components provide data path functionalities. Classical data path functionality, for example, is payload dependent packet filtering, counting or even

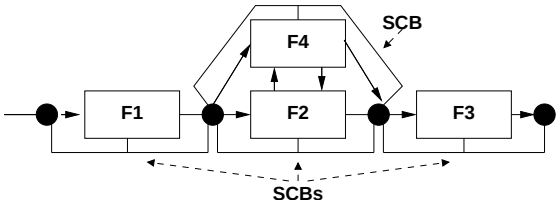

Fig. 1. Service Model

payload transcoding. Service components are configured and controlled by control components. A control component, as exemplified by F4 in Fig. 1, may control one or more service components. In addition, the control component itself may provide data path service functionality. Functionalities provided by the components depend on their implementation. While service components register for data communication only, a control component may register for timed events, too. At vertices, a service graph may be split into several subgraphs and combined by fork and join operations, respectively. Service components register with one data input and one data output port and bidirectional control communication is provided between the service and the relevant control components (symbolized by the connections between F4 and F2).

We refer to service components with the adjacent vertices by the term *service chain*. A *Service Control Bus* (SCB) accompanies a service chain. It propagates signals like packet discard notifications as well as the state of the service chain, i.e. whether the service chain is currently processing a packet or whether it is idle.

3.2 Architecture of PromethOS NP

Fig. 2 depicts the architecture of a PromethOS NP node using a three-tier processor hierarchy[3] and a node control layer.

On all tiers, PromethOS NP provides dynamically code-extensible processing environments (PEs). PromethOS NP creates a hierarchical EE by that an interface to the hierarchical EE is provided only via the control layer. Internally, PromethOS NP manages two different types of code-extensible PEs, in which service components can be installed and instantiated. On the GPP cores, the PE is implemented as an extended PromethOS EE [19] (cf. Host Processor Processing Environment in Fig. 2). This PE provides a binary compatible interface to the PromethOS EE. In contrast to the PromethOS EE, that runs on a single processor node only, the other PE (cf. Network Processor Processing Environments in Fig. 2) is embedded in the hierarchical router platform and provides

[3] The current implementation creates a three-tier hierarchical router platform for nodes that are built of host processor and NP-blades. NP-blades consist of a control processor with a set of packet processors (Appl. Ref. Board [27] for the IBM PowerNP 4GS3 [14]).

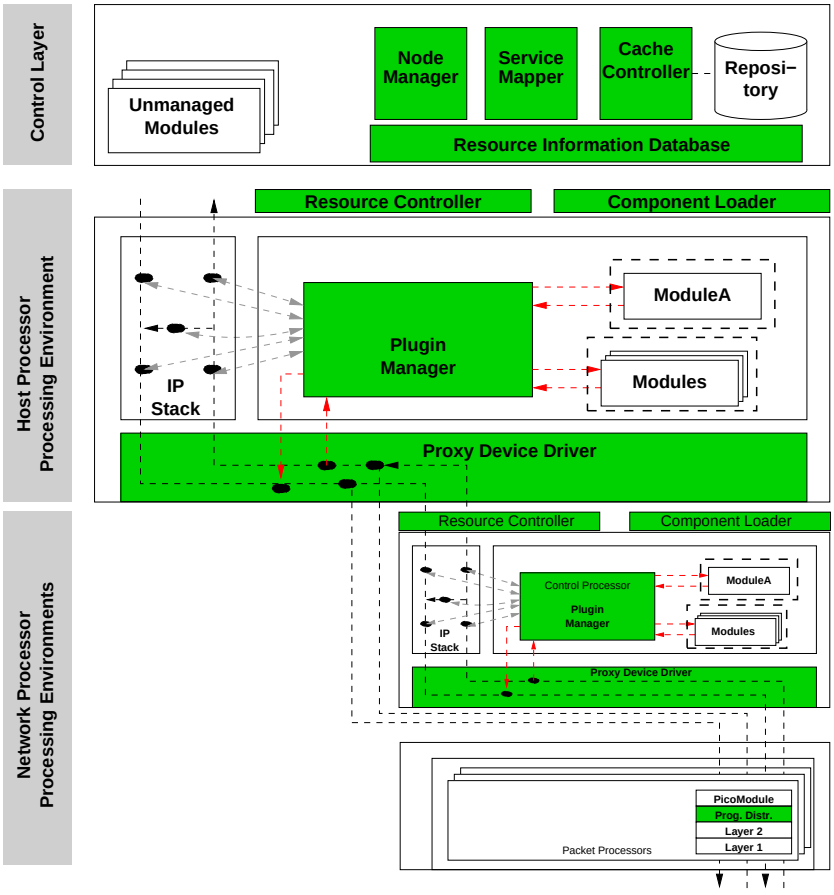

Fig. 2. PromethOS NP Node Architecture

the abstractions to build a service of distributed service components residing in other PEs. On the PPs, a PE is instantiated that provides the mechanisms to install and execute service components without stopping the PP.

The control layer contains components which are responsible for the whole node. The *Node Manager* provides the interface to create a service at node runtime and instructs the other components on the node to act according to its decision. The *Service Mapper* creates the required map specification that provides the information to install and instantiate service components on specific processors such that, first, a service can be created and, second, the resources available are not overbooked. It instructs the PE specific *Component Loaders* to load, instantiate, configure and unload service components. To better differentiate between instantiated and uninstantiated service components, we refer to the latter by the term module. Every module is identified by a module identifier (ModID) that is unique for the whole hierarchical network node. The ModID

is used to query and re-configure the module at run-time. Service components to be instantiated are retrieved by help of the *Cache Controller*. It is responsible to manage the node-local repository which contains service components for PromethOS NP nodes. Upon reception of a request, it either compiles service components from source or retrieves a service component in binary format if available. It does not deal with network-wide service component retrieval but assumes the availability of these components in the node-local repository[4]. The *Resource Information Database* is required to keep track of resources available and consumed. Therefore it interfaces with the *Resource Controllers* residing on the different GPPs. Each PE is controlled by its Resource Controller. The Resource Controller configures and controls the *Programmable Distributors* (PDs) according to instructions received from the Node Manager.

PDs implement the vertices of our service model on and between any processors. Hence, they provide the mechanisms to bind a service chain to specific flows. They are PE specific and provide the mechanisms required to forward packets between service components. Two types of PDs are implemented. One that interconnects service components on the same processor, and the second one that interconnects service components residing on different processors.

In Fig. 3, we illustrate the architecture of a PD. PDs consist of a receiving, classifying and forwarding element [24]. While the receiving and forwarding element eliminate the need of a service programmer to deal with the underlying hardware platform, the classifier element is replaceable. It provides the interfaces like common service components but is required to communicate with the receiving and forwarding elements along the SCB by a particular protocol.

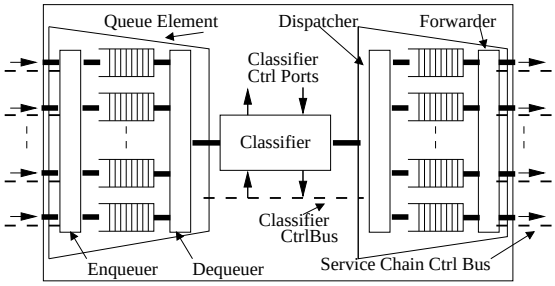

Fig. 3. Programmable Distributor

Packet classification is time-consuming. Therefore, we define cut-through PDs to avoid unnecessary classification overhead if two adjacent components are only linked directly. We extend the basic concept of service chains that consists of a edge between two adjacent vertices to a set of edges for which no

[4] For proof-of-concept purposes, we have implemented a straightforward service component fetcher that is able to retrieve service components from a remote repository over a secured TCP channel if the service components are not locally available.

classification and no inter-processor communication is required in between. On a PromethOS NP node, service chains are identified by the first ModID that starts the chain. A service chain is hooked to multiple outbound ports of a PD as well as multiple different service chains are attached to a PD if required. Dynamic replacement of service chains is based on a selector logic per outbound port. This logic provides the required semantic to install, replace and remove service chains at node run-time without disrupting other services. PDs on PPs are bound to the capabilities offered.

Our *Proxy Device Driver* provides the mechanisms to communicate between all processors of the processor hierarchy[5]. This Proxy Device Driver supports two types of communication channels between different processors. A fast path provides the mechanisms to interconnect service components without additional legacy classification overhead of the Linux Netfilter network stack architecture, while a slow path along the Linux network stack provides the full flexibility of iptables as described for PromethOS [19]. The *Plugin Manager* interfaces with the Linux *IP Stack*, as well as with the fast path. Based on its ability, service components may be executed on nodes with and without NP tiers. Moreover, PDs are implemented as part of the Plugin Manager on GPPs regarding the receiving and forwarding elements.

4 Anti-DDoS Service

Counteracting DDoS attacks requires continuous traffic observation and, if necessary, the installation of countermeasures. Traffic observation and the insertion of countermeasures, however, should not affect regular services. Therefore, we propose a service architecture hereafter that provides the basis infrastructure for the deployment of attack specific functionalities that mitigate the effect of the attacks. The architecture has been designed such a way to make it possible to instantiate the four fundamental functional elements, namely Capture, Identification, Filter and Slowdown (cf. Sec. 2), of our Anti-DDoS service.

4.1 The Service Architecture

Fig. 4 visualizes our Anti-DDoS service architecture in a particular configuration that consists of a basis *service infrastructure* and an attack specific *Service Handler*. While the Service Handler must make the required functionalities available to detect and counteract DDoS attacks, the other components are generic in the sense that they provide the fundamental service architecture. Since the path via the Service Handler creates the needed countermeasure functionality, we refer to this path as the *service path*. Irrespective of the functionality provided, for the PromethOS NP router platform service components are black boxes. As such

[5] Our Proxy Device Driver is based on the code delivered with the IBM Advanced Software Offering Toolkit. It extends the original code base by a generalized, more abstracted communication infrastructure with resource control mechanisms for a hierarchical router architecture built of a multitude of NPs.

not only the service path but also the service infrastructure are built of service components that provide the appropriate functionalities. The service specification is used by the Node Manager that triggers the installation and instantiation of the service as mentioned above. The service logic, however, is service specific. As such, the service logic may contain mechanisms to request the installation or removal of service components depending on service-internal policies. Due to this autonomous, policy based service-internal management, our service architecture provides the basis of a node-local *autonomous service*.

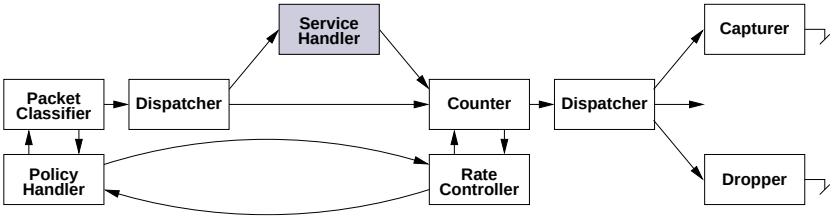

Fig. 4. Anti-DDoS Service Architecture

We argue that this autonomous service provides a suitable basis for detection mechanisms and countermeasures against well-known and unknown attacks. We exemplify three different, particular service configurations to illustrate the applicability of our service architecture for the mechanisms and measures introduced in Section 2:

- Without detection mechanisms or countermeasures installed, the *Packet Classifier* assigns a tag to the incoming packets and sends the packet to the first *Dispatcher*. Since no particular service path is specified by the tag, the Dispatcher forwards the packet to the *Counter*. The Counter increments tag-dependent counters and sends the packet to the next Dispatcher. This Dispatcher then re-inserts the packet into the common routing/forwarding path of the router.

- In case of a well-known attack, whose packets are classified according to specified criteria, our Anti-DDoS service with the detection and countermeasure mechanisms are implemented in the following way. An appropriate policy is given to the *Policy Handler* that creates the service path and configures the Packet Classifier implementing the Capture service function. Policies are specified beforehand and sent to the Policy Handler by service-external entities. Packets matching the criteria are sent to the respective service path (Identification). Service Handlers provide the mechanisms required to detect packets that belong to an attack. Their mechanisms give the specific operations necessary for in-depth payload inspection or multi-protocol attack handling to detect, for example, the W32/Blaster worm [6]. Service Handlers are installed on request to carry

out the Filter or Slowdown service functionalities, as well. Multiple Service Handlers may exist simultaneously. A Service Handler signals the detection of a particular pattern to the subsequent Dispatcher. Depending on the service configuration, packets are sent to the *Capturer* or to the *Dropper*.

– In case of unknown attacks, our Anti-DDoS service follows a different configuration. These attacks, i.e. traffic anomalies, are detected by that the *Rate Controller* queries the counters periodically and compares the values retrieved with specified thresholds. If the counters exceed these thresholds, the Rate Controller informs the Policy Handler of the violating tag and provides the violated condition. This message initiates a service extension or re-configuration process by the Policy Handler. The Policy Handler triggers the installation of a violation specific service path and configures the Packet Classifier to dispatch packets that comply with the specific pattern accordingly. Based on the possibility to extend services dynamically, specific detection mechanisms can be provided to detect and analyze unknown traffic anomalies. The Rate Controller is implemented to control traffic in an autonomous way. Statistical information can provide the means required to detect abnormal traffic patterns. Policies bring the Policy Handler to, for example, configure particular Droppers or Capturers as to implement the Capture, Identification, Filter or Slowdown service mechanisms, respectively.

Packets can be sent to the Service Handler by mistake if, for example, a packet matches a particular byte-pattern at the first classifier but the in-depth packet inspection carried out by the Service Handler reveals that the packet is not part of an attack. Were such packets simply discarded, denial of service results although not all flows are malicious. To avoid such malfunctioning, false positives must be re-inserted.

Attacks vary and provide attack-specific characteristics. These characteristics are yet unknown and may require specific countermeasures that are neither configurable with today's routers nor implementable within today's firmware. Large hierarchical routers located in or close to the core, however, need to be prepared to effectively mitigate future attacks without interruption of other services as it would be required if firmware would need to be upgraded.

4.2 Hardware Constraints

PDs enable service designers to focus on the specific functionality to be implemented according to a unified component model among all types of processors. Thus the challenge remains to decide where to place which service component. For the exploration of this problem space, we need to take hardware constraints into account before we can propose an appropriate classification scheme.

Today's packet processors provide very limited but highly specialized processing capabilities. The programming flexibility known from general purpose processors is not available there. For example, the number of timer events is

small. Since PPs are focused on squeezing out the most of possible performance for packet processing, they are not well-suited for dynamic code updates. Memory is direct mapped; no address virtualization is available. This imposes hard constraints on the code layout of service components for packet processors, and makes the installation of code components at run-time extremely difficult. Fast memory on the NPs is an extremely scarce resource. Different types of memory exist therefore on an NP-blade. Packet processors differentiate between instruction memory and data memory. Often, the former provides room for a total of 32 kilo-instructions [14] only. Thus, the number of code components that can be installed is very limited if we assume that additional functionality, like routing, must be provided by the NP besides our services. While PPs provide fast, co-processor supported packet processing capabilities, control processors on the NPs increase flexibility by general purpose processor architectures. In addition, CPs are able to manage up to 2 GBytes of DRAM [16].

Packet processors are able to forward packets at line speed. But communication paths between service components on the PPs and those on the CPs are not able to cope with the aggregated throughput of all PPs. Neither are today's CPs able to process so high packet rates fully themselves. For example, our prototype implementation with the IBM PowerNP 4GS3, we have been able to receive packets at approx. 100 Mbit/s on the embedded PowerPC. while bi-directional communication resulted in a maximal transmission rate of 42.7 Mbit/s [25][6]. If we assume a hierarchical network node with multiple NP-blades, router-internal communication between NP-blades and the host processor is not able to cope with the data rate either. Actual NPs, like the Intel IXP28xx family, are able to forward packets lossless at rates of up to 20 Gbit/s [17] on the packet processors. Thus, forwarding all packets to host processors would overcharge any of them,

Hence, we argue that scalability of our node is achieved by that programmable network interfaces will be equipped with NPs in the near future. Thus they provide a fully programmable GPP together with a potentially large set of optimized and specialized PPs. Currently, control processors do not provide the processing capacity to run data path service components with the required performance [25]. However, as processor technology advances, performance of control processors will not be of a major concern. We can imagine that multicore CPs are feasible soon as separated memory channels for CPs and PPs are. Separated channels are required such that the processing elements do not interfere with each other[7] when processing packets each. But challenge remains in deciding where to place service components most effectively.

4.3 Service Components on Our Hierarchy-extended Router

Communication between different processor tiers is time consuming, imposes additional limits on packet throughput and comes at the cost of overhead that needs to be avoided by design if possible. To explore this problem space, we

[6] The chip itself is able to handle nearly four times 1 Gbit/s.
[7] Some NP manufacturers provide this capability already today.

propose a classification scheme that is based on the complexity of the operation, the rate and the type of service functionality. This classification scheme is used hereafter to support the mapping strategy of an Anti-DDoS service onto our router platform.

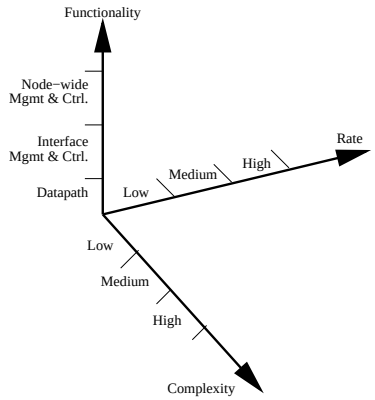

Fig. 5. Classification Dimensions

We differentiate between three dimensions as illustrated in Fig. 5. The first dimension is the complexity of the function provided by the component. The complexity depends on the type of procedures to be applied. For example, if byte-patterns can be identified in a single packet only, the complexity of the appropriate function is lower than if a series of packets needs to be kept in memory before the function can return its decision. The second dimension is the rate a service component must be able to receive packets or control messages. And the third dimension is the type of service functionality provided, i.e. data path, control or management functionality. For the PromethOS NP platform, it is important to know if a service component is triggered by a timer event or upon arrival of a packet at its data input port, and if a service component manages and controls rather a full node than a NP-blade only. We name the first dimension processing *complexity*, to the second we refer by the term *rate*, and the third is referred to as *functionality*.

For the classification of the service components of our service architecture, we decide on the complexity, rate and the functionality. The complexity of a function provided by a component is determined at specification time while the initial location of components in the second dimension is based on rough estimations of expected packet rates for data path service components. The location of control and management components in the second dimension depends on the frequency of triggers or queries the component is expected to handle. Placement of components in the third dimension is based on the type of service functionality a component is expected to provide. It is important to distinguish from service components that provide functionality on data packets and from those

that are able to control manage other service components. Hence, the functionality differentiates between service components residing solely in the data path or providing functionality in the control and management plane as well.

Component	Complexity	Rate	Functionality
Classifier	Low	High	Data path
Counter	Low	High	Data path
Capturer	High	Low	Data path
Dispatcher	Low	High	Data path
Dropper	Low	High	Data path
Rate Controller	Low	Middle	Interface Control
Service Handler	Low → High	Low → High	Data path/Interface Control
Policy Handler	High	Low	Node-wide Management & Control

Table 1. Classification

In table 1, we present the classification of the service components of our service architecture according to the scheme introduced above. We exemplify the classification by explaining its application to the Packet Classifier, the Service Handler and the Policy Handler. The Packet Classifier needs to process all arriving packets. Hence, the rate is high. However, the complexity is low since packets can be classified based on data available in a single packet only. Remember that this Classifier decides if a packet needs further in-depth analysis. In addition, the functionality provided is one clear representative of the data path. Classifier are therefore instantiated on PPs of all NP-blades most preferably. Functionality provided by the Service Handler is service specific. The operations to be applied can range from rather simple byte-pattern matching of payload in potentially fragmented IP packets up to complex multi-protocol attack detection, or the detection of commonalities found in traffic anomalies of which the reason is unknown. Thus, its complexity may range from low up to high. The same argumentation applies for the packet rate it must be able to process. Functionality provided by the Service Handler may reside in the data path and/or in the control plane. Since the Service Handler itself can be composed of various components, full flexibility is required of PromethOS NP to install the specific components on appropriate processors. It is important to notice that PromethOS NP distributes internal control messages between different processors on the same mechanisms as used for data communication, and imposes therefore no limitations. Depending on their complexity, service handlers are therefore instantiated either on CPs or PPs of the required NP-blades. The Policy Handler provides management and control functionality that supervises potentially all NP-blades. As an aggregating function with an expected low-bandwidth communication interface but high service complexity, it is predestined to be instantiated in the PE on the host GPP. Following this classification of the components, we specify the following mapping of components on the hierarchy-extended platform:

– Components with a low complexity and a high packet rate are placed on packet processors most preferably. Thus, promising candidates are the Packet Classifier, the Counter, Dispatchers as well as the Dropper. Depending on the complexity of the Service Handler, parts of it like byte-pattern matching are candidates as well.

– Components with a high complexity and a low rate are installed either on the control or on the host processor. The exact placement depends on the kind of interaction among components. Thus, candidates to be installed on these GPPs are the Rate Controller, the Policy Handler and the Capturer. For obvious reasons, particular components of the Service Handler may or must reside on GPPs, as well.

– The functionality dimension determines if a component can be installed on packet processors or must be placed on GPPs. Data path components can be installed anywhere. Although theoretically feasible on PPs, control and management components that are triggered by timer events are installed on GPPs. Thus we decide to place the Rate Controller on the respective CP of the PPs, and place the Policy Handler on the host processor.

PDs provide resource accounting and enforcement mechanisms. By the means of the Resource Controller, service re-configuration can be implemented to allow for the relocation of service components at service run-time. Thus, service components could be relocated if their location in the classification scheme changes. However, our platform provides no explicit support for state preserving or service component migration. Based on the measurements of the Rate Controller, the Policy Handler could provide the required functionalities to trigger a redeployment of the service with different attributes.

5 Related Work

Various active router platforms following the component model have been proposed for single processor systems [1, 10, 20]. However, only a few addressed the problem of managing hierarchical active network nodes with integrated support of NPs.

VERA [18] introduced the hierarchy of classifiers as a chain of classifiers which is mapped on a model of a hierarchical router. It defines extensibility as the ability to provide resources for additional services. However, the core components of VERA do not provide at run-time extensibility, and VERA does not deal with the complexity of instantiating services that span all tiers arbitrarily. Compared with PromethOS NP, VERA takes programmability of packet processors into account, but packets are forwarded to a statically linked operating system running on the host GPP.

NetBind [4] proposes an approach to construct data paths dynamically on a network processor based router. Low latency on dynamic binding is achieved

due to post-processing of intermediary object files before linking the components. For this reason, no overhead takes place at the execution time, except for the machine code changes. In comparison to PromethOS NP, NetBind is not a generic framework for adding new services on network processor based routers, i.e., NetBind does not deploy services on all tiers of the processor hierarchy in an integrated way.

SPLITS [12] creates a router architecture built of attached network processors line cards and host processors. While SPLITS provides the same functionality as VERA and NetBind for network processors, functionality is extended by stream handlers that allow for flexible interception of packet flows to attach arbitrary applications. Like VERA and NetBind, SPLITS does not address the potential of CPs for the execution of services. However, we are convinced CPs are and will be an important processing element on large hierarchical routers for router scalability reason. Therefore, we provide the concepts and mechanisms required for services of which service components reside on one, several or all processor tiers including CPs.

The potential of active countermeasures against large-scale distributed denial of service attacks in the Internet has been recognized before. FIDRAN [13] proposes a service framework similar to ours. However, the service architecture focuses on a single host GPP node only, and hence, is not able to benefit from additional processing capabilities offered by a hierarchy extended active network node. FLAME [2] built and evaluated a monitoring system that can be used to detect distributed denial of service attacks. Similar to FIDRAN, the system is designed for single host GPPs only. In [21], the application of re-configurable hardware to detect signatures in payload of packets is proposed. While we are convinced that the FPX is able to scan packets for signatures much faster than our architecture, we argue that our architecture provides more flexibility as required for, e.g. selective packet capturers.

6 Conclusions and Outlook

In this paper, we have analyzed the problem space of detection mechanisms and countermeasures against large-scale distributed denial of service attacks in the Internet, and presented briefly the architecture of PromethOS NP. PromethOS NP provides a dynamically extensible router platform for hierarchical network nodes built of host and network processors for high-performance packet processing. Motivated by the continuously increasing significance of the Internet to business and commerce, and the always quicker spreading of newly created worms and viruses, we have proposed a service architecture that allows for the efficient deployment of new service functionalities to detect and counteract DDoS attacks effectively on high-performance routers for the Internet backbone. The classification scheme proposed in this paper alleviates the design and implementation of specific Anti-DDoS service components that benefit from our service architecture, as well as from the flexibility and the capabilities of our PromethOS NP router platform. Hence, we are convinced that our service architecture in

conjunction with PromethOS NP provides the flexibility and performance required for detection mechanisms and countermeasures against DDoS attacks in the Internet. Moreover, it is flexible enough to provide the basis for services in completely different fields like charging and accounting of traffic. Currently, we are implementing the proposed service architecture on PromethOS NP. The evaluation of this implementation with appropriate Anti-DDoS service components will show whether our claims hold.

References

1. D.S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith. The SwitchWare active network architecture. *IEEE Network*, 12(3), May/Jun. 1998.
2. K.G. Anagnostakis, S. Ioannidis, S. Miltchev, J. Ioannidis, M.B. Greenwald, and J.M. Smith. Efficient packet monitoring for network management. In *Proc. of IFIP/IEEE Network Operations and Mgmt. Symp. (NOMS) 2002*, Apr. 2002.
3. CAIDA. CAIDA Analysis of Code-Red. http://www.caida.org/analysis/security/code-red, 2003.
4. A.T. Campbell, M.E. Kounavis, D.A. Villela, J. Vicente, H.G. de Meer, K. Miki, and K.S. Kalaichelvan. NetBind: A Binding Tool for Constructing Data Paths in Network Processor-based Routers. In *Proc. of the 5th Int. Conf. on Open Architectures and Network Programming (OPENARCH)*, Jun. 2002.
5. CERT. CERT Advisory CA-2003-04 MS-SQL Server Worm. http://www.cert.org/advisories/CA-2003-04.html, 2003.
6. CERT. CERT Advisory CA-2003-20 W32/Blaster Worm. http://www.cert.org/advisories/CA-2003-20.html, 2003.
7. Cisco. White Paper: NetFlow Services and Applications. http://www.cisco.com, 2002.
8. Ferngesteuerte Spam-Armeen. c't Magazine, Issue 5, 2004.
9. R. Danyliw and A. Householder. CERT Advisory CA-2001-19 "Code Red" Worm Exploiting Buffer Overflow in IIS Indexing Service DLL. http://www.cert.org/advisories/CA-2001-19.html, 2001.
10. D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner. Router plugins: A software architecture for next-generation routers. *IEEETNWKG: IEEE/ACM Trans. on Networking. IEEE Comm. Society, IEEE Computer Society and the ACM with its Special Interest Group on Data Comm. (SIGCOMM), ACM Press*, 8, 2000.
11. T. Dübendorfer, A. Wagner, and B. Plattner. An Economic Damage Model for Large-Scale Internet Attacks. In *13th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WET ICE 2004); Workshop on Enterprise Security, Modena, Italy*, 2004.
12. A. Gavrilovska. SPLITS Stream Handlers: Deploying Application-level Services to Attached Network Processors. Ph.D. Thesis, Georgia Institute of Technology, Jul. 2004.
13. A. Hess, M. Jung, and G. Schäfer. FIDRAN: A Flexible Intrusion Detection and Repsonse Framework for Actie Networks. In *Proc. of the 8th IEEE Symp. on Computers and Communication (ISCC), Kemer, Antalya, Turkey*, Jul. 2003.
14. IBM Corp. IBM PowerNP NP4GS3 databook. http://www.ibm.com, 2002.
15. Intel Corp. Intel IXP1200 Network Processor – Datasheet. http://www.intel.com, 2000.

16. Intel Corp. Intel IXP2800 Network Processor Hardware Reference Manual. http://www.intel.com, Nov. 2002.
17. Intel Corp. IXP2800 Intel Network Processor IP Forwarding Benchmark Full Disclosure Report for OC192-POS. http://www.intel.com, Oct. 2003.
18. S. Karlin and L. Peterson. VERA: An extensible router architecture. In *Proc. of the 4th Int. Conf. on Open Architectures and Network Programming (OPENARCH)*, Apr. 2001.
19. R. Keller, L. Ruf, A. Guindehi, and B. Plattner. PromethOS: A Dynamically Extensible Router Architecture Supporting Explicit Routing. In *Proc. of the 4th Annual Int. Working Conf. on Active Networks IWAN, Zurich, Switzerland*, number 2546 in LNCS. Springer, Dec. 2002.
20. E. Kohler, R. Morris, B. Chen, J. Jannotti, M. Kaashoek, and C. Modular. The click modular router. *ACM Trans. on Computer Systems*, 18(3), Aug. 2000.
21. J. W. Lockwood, J. Moscola, D. Reddick, M. Kulig, and T. Brooks. Application of Hardware Accelerated Extensible Network Nodes for Internet Worm and Virus Protection. In *Proc. of the 5th Annual Int. Working Conf. on Active Networks IWAN, Kyoto, Japan*, number 2982 in LNCS. Springer, Dec. 2003.
22. Network Processing Forum. http://www.npforum.org, Jun. 2004.
23. J. Postel. Transmission control protocol. RFC 792, ISI, Sep. 1981.
24. L. Ruf, R. Keller, and B. Plattner. A Scalable High-performance Router Platform Supporting Dynamic Service Extensibility On Network and Host Processors. In *Proc. of 2004 ACS/IEEE Int. Conf. on Perv. Services (ICPS'2004)*. IEEE, Jul. 2004.
25. L. Ruf, R. Pletka, P. Erni, P. Droz, and B. Plattner. Towards High-performance Active Networking. In *Proc. of the 5th Annual Int. Working Conf. on Active Networks IWAN, Kyoto, Japan*, number 2982 in LNCS. Springer, Dec. 2003.
26. S.E. Schechter and D.S. Smith. Access For Sale. In *ACM Workshop on Rapid Malcode (WORM)*, 2003.
27. Silicon Software System. Application Reference Board for the IBM PowerNP NP4GS3 Network Processor User Manual. http://www.s3group.com, 2002.
28. K. Sollins. The TFTP Protocol (Revision 2). RFC 1350, MIT, Jul. 1992.
29. S. Staniford, V. Paxson, and N. Weaver. How to Own the Internet in Your Spare Time. In *Proc. of the 11th USENIX Security Symp.*, Aug. 2002.
30. D. Tennenhouse and D. Wetherall. Towards an Active Network Architecture. In *Multimedia Computing and Networking (MMCN 96), San Jose*, 1996.
31. The Economist. E-commerce takes off. *The Economist*, 371(8375):9, May 2004.
32. A. Wagner, T. Dübendorfer, B. Plattner, and R. Hiestand. Experiences with Worm Propagation Simulations. In *ACM Workshop on Rapid Malcode (WORM)*, 2003.
33. A. Wagner and B. Plattner. Peer-to-peer systems as attack platform for distributed denial-of-service. In *ACM SACT Workshop, Washington, DC, USA*, 2002.