Improving Traffic Classification and Policing at Application Layer

Helder Pereira and Andre Ribeiro
Portugal Telecom Inovação, SA
3810-1/6 Aveiro, Portugal
Email:{helder-m-pereira,andre-g-ribeiro}@ptinovacao.pt

Paulo Carvalho
University of Minho, Department of Informatics
4710-057 Braga, Portugal
Email:pmc@di.uminho.pt

***Abstract*—The common process of classifying network traffic resorting to a set of IP header fields and well-known communication ports is highly fallible as some applications try to hide their true nature by, for instance, using dynamic, non default ports. In this paper, we argue and demonstrate that application layer inspection is a possible and convenient approach to derive the correct application protocol. This detection and classification process is crucial to allow an efficient control of traffic entering the network. Taking pfSense as a case study, we extend its current layer 3 and 4 classification scheme with layer 7 (L7) capabilities, providing a powerful solution to control traffic based on application patterns. We propose the concept and use of L7 containers so that a user can easily create a set of rules for inspection, which will drive lower-level traffic control. In addition, we propose and implement a mechanism to create automatically useful application inspection scenarios.**

I. Introduction

The integration of a vast range of applications and services in a common network communication infrastructure as the Internet stresses the need to perform efficient traffic classification and policing, either due to resource management, charging or security concerns. The traditional way to classify traffic entering a network domain is usually based on network and transport data fields, e.g. service class marks, source and/or destination IP addresses and ports. Although in many cases this type of classification offers a good compromise between simplicity and efficacy, it fails to address particular cases where those fields are somehow inconclusive or unavailable. In fact, classification at layer 3 and 4 is hard to archive in presence of peer-to-peer (P2P) traffic as the applications use a range of random, non default ports. The typical cases of a HTTP server running on different ports from port 80 (port hopping) and encrypted connections are examples impairing proper classification. In this context, performing traffic classification and policing at the application layer (L7 in short) can be a convenient solution to overcome these limitations. In L7 classification, user traffic can be identified based on an application pattern. An application pattern is a sort of signature used by an application during its communications. Examples of related work on L7 classification include IPCop Firewall [1] and Bandwidth Arbitrator [2].

In the present work, we study and tackle the L7 classification paradigm for the pfSense platform [3]. In this context, we have established the following goals: i) to develop mechanisms to control the classification component in the application protocol, integrating them in the pfSense platform through a graphical interface; ii) to define and implement user-friendly wizards to simplify the configuration of QoS rules; iii) to plan and develop a test platform which allows testing multiple patterns of applications simultaneously, and to measure the performance (e.g. response time) of the classification module based on the application layer.

The contents of this paper is organized as follows: the pfSense platform is briefly described in Section II; the design and implementation issues regarding the L7 classification solution are explained in Section III; the proof-of-concept is included in Section IV; and finally, the main conclusions are summarized in Section V.

II. The pfSense platform

A. A brief description

pfSense is a customized FreeBSD distribution, mainly oriented to be used as a firewall and router [3]. pfSense uses a single XML file, called *config.xml*, that stores the configuration of all services available in the pfSense host. This allows pfSense to be easily backed up, since *config.xml* is a self-contained configuration file, i.e. a newer machine can be fully constructed from scratch with a single file restore.

QoS management in pfSense is achieved through the AltQ framework [4]. AltQ is used to provide queuing disciplines and other QoS mechanisms in order to perform resource sharing and QoS control. In the newest builds (2.0 builds), pfSense includes an additional shaping mechanism, called Dummynet [5]. Dummynet was originally available to the ipfw firewall, but recently the pf firewall (which is used in pfSense) is able to use it. At present, QoS management in pfSense is carried out at the layer 3 and layer 4 of the OSI model. This means that applications and services are only identified by IP header fields and/or by a set of standard ports. This is a major limitation as many software applications do not use well-known ports to communicate, suffer from “port hopping”, or may change its communications port at any time in an unpredictable way.

As mentioned before, performing traffic classification at the application layer may be a prominent answer to address this problem.We will address this issue and study how to integrate L7 inspection capabilities within pfSense.

III. L7 Traffic Control: Design and Implementation

A. Designing the Solution

The initial design goal behind our L7 traffic control solution was to balance appropriately the granularity and simplicity of the configuration process. When exposing L7 features to the end user, it is convenient to provide detailed configuration facilities in order to allow a fine tuning of control options, but also to give the opportunity to create a set of pre-defined parameters in a simple way, based on a set of pre-defined options. With these two goals in mind, the design options and components for the proposed solution are discussed next.

L7 classification is performed using an application called *ipfw-classifyd*. This application is able to: (i) produce blocking rules for incoming traffic; or (ii) perform traffic shaping by assigning IP packets of a specific flow to an AltQ queue or to a Dummynet pipe or queue. For simplicity, from now on, the term "structure" is used to refer either to an action, an AltQ queue or a Dummynet pipe or queue; the values of the structures themselves, such as "block", will be called "behaviors". A more in-depth explanation of how *ipfw-classifyd* works will be given in Section III-B.

An important goal is to provide a graphical user interface in order to allow the user to leverage the potentialities of *ipfw-classifyd*. Here, the trade-off between configuration simplicity and granularity has to be taken into consideration.

For the sake of simplicity, the user should not be aware of the framework names that will be used in the structures. Therefore, instead of displaying the structures to the user as AltQ, Dummynet pipe and Dummynet queue, a simple and intuitive terminology should be used, hiding the underlying framework. Thus, the user sees as available structures: (i) Action; (ii) Queue (which contains AltQ queues); and (iii) Limiter (which contains Dummynet pipes and Dummynet queues). These simplified names were chosen in accordance with terminology of other pfSense shaping mechanisms. As regards the Dummynet components in the Limiter structure, it was decided that the user does not need to know the difference between a pipe and a queue, being that difference automatically detected by the back-end code.

Establishing an arbitrary number of protocol definitions is a relevant option for the user. However, duped definitions, such as defining the same protocol twice, must not be allowed.

When L7 rules definition is created, the user shall have a way to assign that rule to a pf rule. Due to a current *ipfw-classifyd* limitation, the user will not be able to apply L7 inspection rules to other traffic than TCP/UDP.

Other important design goal is to provide an easy way to create a set of pre-defined L7 rules. This means that during the creation of the rules, the user options are restricted to the selection of protocols.Structures and behaviors should be provided automatically in accordance with the classification group to which a specific protocol belongs to. In this context, the decision of changing the existing traffic shaper, extending it with L7 rules was considered the best option.

B. Implementation

The steps toward the implementation of the proposed solution are detailed in this section. To better understand our proposal, some knowledge is required on how the pfSense framework exposes its features and how the related tools work.

The implementation phase can be generically divided in four steps: (i) L7 classification - involves detailing the inner working components of *ipfw-classifyd*, how to leverage it, understanding the AltQ framework and the Dummynet application; (ii) traffic redirection - involves redirecting traffic that matches a pf rule to *ipfw-classifyd* in order to enforce the behavior defined by L7 rules; (iii) definition of rules - involves the creation of a graphical web interface where L7 rule definitions can be built, specifying what to do when there is a match with a specific application protocol. These rule definitions, from now on will be called L7 *containers*; (iv) definition of wizards - involves the implementation of a wizard where a L7 container with a pre-defined set of L7 rules can be easily created, through few simple steps, in an automated and transparent way to the end user.

1) ipfw-classifyd: As stated before, *ipfw-classifyd* is the application responsible for L7 classification. *ipfw-classifyd* has a straightforward configuration. Essentially, it allows three different types of operations to be applied to an identified application protocol, namely, defining an action to apply, typically a block action, assigning traffic to an AltQ queue, or to a Dummynet pipe or a Dummynet queue. This allows some granularity, however, knowing that AltQ queues and Dummynet pipes and queues are defined in the pf rules file, it would not be easy to specify them clearly within the *ipfw-classifyd* file in a non automatized way.

A way to expose the *ipfw-classifyd* features to the user, while abstracting the QoS mechanisms and their implementation details, is therefore recommended. In order to expose *ipfw-classifyd* capabilities, the concept of *L7 containers* was introduced and implemented. A L7 container is a structure that contains multiple definitions for different application protocols. It allows to specify what to do with each protocol the user intends to configure.

As mentioned before, there are three different types of operations that can be applied to an identified protocol. These type of operations are a miscellaneous of *Structures* and *Behaviors*. A *Structure* can be an "Action", a "Limiter" or a "Queue". A "Limiter" corresponds to a Dummynet Structure and a "Queue" to an AltQ queue. A *Behavior* is related to a structure. For the Structure Action, the only available behavior is "block"; for the Limiter, it is a set of Dummynet pipes and queues; and for the Structure Queue, the behavior is a set of AltQ queues. The behaviors for Queue and Limiter must be present in the system (i.e. they should have been already created in the right places). After the user defines a L7 container, all data is there to produce a *ipfw-classifyd* configuration file successfully. However a way to send the traffic to *ipfw-classifyd*, is still needed. In this context, pf recently acquired one more option, called divert sockets.

2) Divert sockets: Divert sockets are, essentially, a way for the kernel to send network traffic to the user context. As regards *ipfw-classifyd*, diverting actually interrupts the normal flow of packets and sends them to a listening socket (in this case, *ipfw-classifyd*), or sends data directly to the IP processing routine. This is a context switch, in which packets abandon the kernel and go to userland. Clearly, this context switch has an overhead attached to it. To keep it low and controlled to minimize the impact on performance, *ipfw-classifyd* takes some actions. First, if pf is taught previously about the actions to take, when *ipfw-classifyd* daemon decides that something should change from its normal workflow, the effects of context switching can be reduced. The overhead can also be reduced with the definition of a pf rule option to limit the number of packets that are diverted from the kernel to the user software application. In a first approach, and for test purposes, a maximum of five packets are diverted (max-packets=5). This is an experimental value and in the future it will be user definable in order to allow fine tuning of the classification mechanism.

The need to tell pf in advance what actions to take, also requires that pf knows the corresponding structures (action, AltQ, Dummynet) to overload. This indication is given through the "keep state" option of a pf rule, and written as "overload structure diverttag". After this, traffic needs to be diverted to ipfw-classifyd. Again, for test purposes, ports in the 40000-60000 range were considered for *ipfw-classifyd*.

3) L7 container interface: This section describes the proposed graphical interface which allows the user to create and change L7 containers that dictate ipfw-classifyd behaviour. A view of the graphical web interface for creating the rule containers is depicted in Figure 1. As regards the interface design and implementation, several relevant decisions are highlighted.

Each container may have more than one application protocol specified, however, application protocols cannot have dupe specifications. In each rule that is created, the "protocol", "structure" and "behavior" must be specified. The "protocol" field allows the user to choose the protocol application to create a rule for; the "structure" and "behavior" are an aggregate pair conditioning what to do with the detected traffic. The

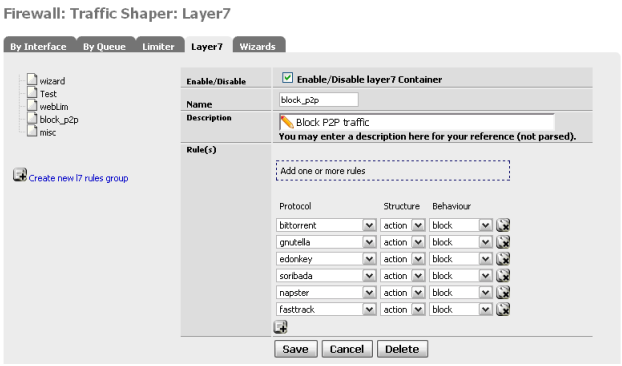

Fig. 1. Graphical interface for creating rule containers

"structure" field is also dynamically filled only with structures having, at least, one queue or pipe defined.

4) Assigning a L7 container to a traffic flow: For each container that is built, the user may decide to assign it to a firewall rule. To cover this facility, a specific field was inserted into the Firewall Rules Edit page so that the user may take that option. If a L7 container is chosen to be applied, all traffic matching that rule will be analyzed by *ipfw-classifyd*.

Due to a current limitation of *ipfw-classifyd*, only UDP and/or TCP streams can be diverted to *ipfw-classifyd*. Improvements to *ipfw-classifyd* are underway as it is currently a work-in-progress. In addition, for each firewall rule, only one L7 container can be assigned due to a pf limitation.

5) Creating L7 aware wizards: At this point, a straightforward way of allowing users to create a standard set of rules was still missing. As pfSense platform had already wizards to configure the shaper, our first thought was to create an explicit L7 configuration wizard. After some discussion it was decided that the current wizards should be extended to transparently include L7 capabilitiesIn a later stage, the solution shall migrate to a specific L7 wizard.

All AltQ queues created by the wizard are used in the L7 configuration file in order to mimic non L7 shaper behaviors in *ipfw-classifyd*.

As regards VoIP services and applications, it was decided that the most common application protocols related to VoIP would be assigned to a VoIP queue, without showing this to the end user. This is completely transparent to the user, to whom the only concern is selecting what application protocols to shape. If the traffic to shape results from Peer-to-Peer applications, the select box on top of the page can be enabled and the related protocols or applications can be selected (blocked) one by one. By default, the wizards (P2P, network gaming, etc.) already comprise a comprehensive set of application protocols.

Although the application protocol verification is currently performed by port and pattern, the objective is to become only pattern-based. This improvement will be done in the near future.

6) Upgrading or adding new L7 pattern files: A relevant feature to improve the support for L7 inspection is the possibility of allowing the user to upgrade the platform with new application protocol patterns. This feature is important when the user wants to block an application that uses a protocol pattern that is not defined in the system. If a new pattern is uploaded to the system, it only appears in the list of protocols when a container is created or modified. The defined wizards are not affected.

IV. Results

In this section, the process of L7 classification and policing is discussed from a practical perspective, highlighting representative configuration steps and obtained results.

We will show the result of creating a simple L7 container with a strict block policy (block_p2p) and how it is stored in *config.xml*. Definitions from the L7 container GUI are easily

stored in *config.xml*. As there is only a block policy, the translation from the *config.xml* definition to the *ipfw-classifyd* configuration file[1] is fairly strict. This is illustrated in the following *ipfw-classifyd* configuration file extract:

```
bittorrent = action block
edonkey = action block
fasttrack = action block
```

Next, this container is assigned to a specific Firewall Rule,As there is only a single block policy, the resulting pf rule is also simple:

```
pass  in  quick  on $LAN  proto { tcp udp }
from {  192.168.160.2 } to 192.168.87.2 divert 47244
keep state ( max-packets 5, overload action
diverttag ) label "USER_RULE: Layer7 block P2P"
```

It is clear that the overload option is correctly created, since only one action needs to be overloaded. As illustrated, the creation of a L7 container is straightforward and works as expected. The other structures were also tested and they work as expected, producing correctly formatted rules. As an example, Figure 2 illustrates a container including the three structures configured.

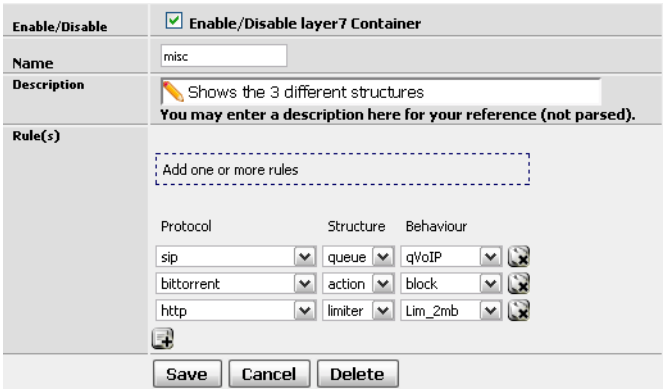

Fig. 2. Miscellaneous container creation

Next, it will be shown how the shaper wizard was implemented. As stated before, the rules produced by the wizard are completely transparent to the final user. The user only selects the applications where shaping is due to be applied, and the wizard is smart enough to understand which application protocol is related with each particular application. Since this is a shaping policy, only AltQ queues are assigned. Several different queues with specific QoS parameters, adapted to the expected type of traffic, can be observed. The QoS parameters for these queues are automatically derived from the link bandwidth and the type of traffic they are going to receive. Part of the wizard *ipfw-classifyd* configuration file is shown below:

```
sip = queue qVoIP
...
bittorrent = queue qP2P
edonkey = queue qP2P
...
xboxlive = queue qGames
doom3 = queue qGames
...
rdp = queue qOthersHigh
vnc = queue qDefault
msnmessenger = queue qOthersLow
...
```

The pf rule responsible for diverting traffic to *ipfw-classifyd* is slightly different from the other ones. This is a directionless rule that is automatically created by the wizard, and known in the pfSense terminology as a "Floating Rule". As the wizard only assigns AltQ queues to the different application protocols, only AltQ needs to be overloaded in the pf rule. The resulting pf rule for this L7 container is the following:

```
pass out proto { tcp udp } from any to any
divert 50476 keep state ( max-packets 5, overload
altq diverttag ) label "USER_RULE: Layer7 wizard"
```

Once again, the results are consistent with the envisioned system design, where this rule will enforce shaping of traffic going through that particular pfSense box.

V. Conclusions

We have proposed and implemented a powerful and flexible approach to perform traffic classification and policing at the application level. This approach addresses the potential inefficiency of layer 3 and 4 classification, being an effective solution to overcome it. The solution, integrated seamlessly into pfSense, has been enhanced with a useful GUI that allows the end user to leverage L7 capabilities that *ipfw-classifyd* provides. Through the use of L7 containers, the user is allowed to create a set of rules for L7 inspection, as a complement to traditional traffic classification schemes. In addition, specific wizards have been defined to automate control of typical applications and services such as VoIP, P2P and network gaming.

Acknowledgements

The authors would like to thank to Ermal Luçi all the precious help he gave during the course of this work, specially, regarding ipfw-classifyd and pf. The authors also would like to thank Scott Ulrich and Chris Buechler, founders of pfSense, for their feedback and guidance during the course of the project, as well as, the active pfSense developers for their ideas and opinions.

References

[1] IPCop Firewall. URL: http://www.ipcop.org, 2003.
[2] Bandwidth Arbitrator. URL: http://www.bandwidtharbitrator.com/, 2002.
[3] pfSense Project. URL: http://www.pfsense.com/, 2004.
[4] Kenjiro Cho. A framework for alternate queueing: towards traffic management by pc-unix based routers. In *ATEC '98: Proceedings of the annual conference on USENIX Annual Technical Conference*, Berkeley, CA, USA, 1998. USENIX Association.
[5] Marta Carbone and Luigi Rizzo. Dummynet revisited. *SIGCOMM Comput. Commun. Rev.*, 40(2):12–20, 2010.

[1] As *ipfw-classifyd* is currently work-in-progress and is evolving at a good pace, some of the files or configurations included in this paper may suffer minor changes.