Scalable Rule Management for Data Centers

Aris Paphitis
CUT

Summary Of

Summary of the article. Keypoints, contributions and comments.

1 Background

Cloud operators increasingly need more and more fine grained rules to better control individual network flows for various traffic management policies. Many novel traffic policies have been proposed by recent research, to improve network utilization, application performance, fairness and cloud security among tenants in multi-tenant data centers.
Tenants in data centers like Amazon or whose servers run software from VMware, place their rules at the servers that source traffic. However, multiple tenants at a server may install too many rules at the same server causing unpredictable failures. Given the scale of today's data centers, the total number of rules can be hundreds of thousands or even millions. In perspective, rule processing of future data centers can hit CPU or memory resource constraints at servers and switches.

2 Motivation

In this paper, the authors argue that future data centers will require automated rule management in order to ensure rule placement that respects resource constraints, minimizes traffic overhead, and automatically adapts to dynamics. They propose vCRIB, a virtual Cloud Rule Information Base, which provides the abstraction of a rule repository, and automatically manages rule placement without intervention by operator. To achieve this they propose offloading rule processing to other devices in the data center.

3 Challenges

The authors acknowledge the need for many fine-grained rules to achieve different management policies; typical examples are data centers that provide computing as a service. They observe that a simple policy can result in a large number of fine grained rules, especially when operators wish to control individual VMs and flows. Additionally, in a data center where multiple concurrent policies might co-exist, rules may have dependencies between them, so may require carefully designed offloading. vCRIB is focused on flow-based rules (drop, rate limit, count) that can be processed either at servers (hypervisors) or programmable network switches. These rules are usually based on IP addresses, MAC addresses, ports, VLAN tags etc. While software-based hypervisors at servers can support complex rules and actions, they may require committing an entire core at each server in the data center. Some hypervisors may offload rule processing to the NIC, but it can only handle limited number of rules due to memory constraints.

Rule management is further complicated by heterogeneity in hardware equipment and the highly dynamic environment with policy changes existing in today's data centers.

The central challenge of vCRIB is to design a collection of algorithms that manages to keep the traffic overhead induced by rule offloading low, while respecting the resource constraints.

4 Contributions

To address these challenges vCRIB provides an abstraction of a centralized repository of rules for the cloud. Tenants and operators simply install rules in this repository and vCRIB uses network state and traffic information to proactively place rules in hypervisors.

4.1 System Design

vCRIB partitions the rule space to break dependencies among rules. Each partition contains rules that can be co-located. Rules that span multiple partitions are replicated rather than split; this reduces rule inflation. In vCRIB this is called `rule partitioning with replication` and it allows a flexible and efficient placement of rules at both hypervisors and switches. Different types of rules may be best placed at different places. vCRIB uses `per-source partitions` such that within each partition, all rules have the same VM as the source. This way only a single rule is required to redirect traffic when that partition is offloaded. Similarities in co-located rules are treated with care not to double resource usage. vCRIB uses resource usage functions that deal with different constraints (CPU/memory) in a uniform way, accommodating device heterogeneity. The process of rule offloading is split in two steps: a novel bin-packing heuristic for `resource-aware partition placement` which leverages similarity and respects resources and a fast `online traffic-aware refinement` algorithm which migrates partitions to reduce traffic overhead. The split enables quick adaptation to small scale dynamics. Finally, the resource-aware placement of rules is achieved through an heuristic algorithm called `First Fit Decreasing Similarity` and is optimized via a `traffic-aware refinement` to reduce traffic overhead while still maintaining feasibility, in the sense that it respects resource constraints.

5 Evaluation

vCRIB's ability to minimize traffic overhead given resource constraints and rule replacement were studied through simulations on a large fat-tree topology.

Resource usage and traffic trade-off vCRIB uses similarity to find feasible solutions when source placement is infeasible vCRIB finds a placement with low traffic overhead. It can also optimize placement given CPU constraints. Resource usage and traffic spatial distribution vCRIB is effective in leveraging on-path and nearby devices. most traffic overhead introduced is within the rack Parameter sensitivity analysis similarity is used to accommodate larger partitions Reaction to cloud dynamics

More fascinating text. Features[1] galore, plethora of promises.

[1] Remember to use endnotes, not footnotes!

6 This is Another Section

Some embedded literal typset code might look like the following :

```
int wrap_fact(ClientData clientData,
              Tcl_Interp *interp,
              int argc, char *argv[]) {
    int result;
    int arg0;
    if (argc != 2) {
        interp->result = "wrong # args";
        return TCL_ERROR;
    }
    arg0 = atoi(argv[1]);
    result = fact(arg0);
    sprintf(interp->result,"%d",result);
    return TCL_OK;
}
```

Now we're going to cite somebody. Watch for the cite tag. Here it comes... The tilde character (~) in the source means a non-breaking space. This way, your reference will always be attached to the word that preceded it, instead of going to the next line.

7 This Section has SubSections

7.1 First SubSection

Here's a typical figure reference. The figure is centered at the top of the column. It's scaled. It's explicitly placed. You'll have to tweak the numbers to get what you want.

This text came after the figure, so we'll casually refer to Figure 1 as we go on our merry way.

7.2 New Subsection

It can get tricky typesetting Tcl and C code in LaTeX because they share a lot of mystical feelings about certain magic characters. You will have to do a lot of escaping to typeset curly braces and percent signs, for example, like this: "The `%module` directive sets the name of the initialization function. This is optional, but is recommended if building a Tcl 7.5 module. Everything inside the `%{, %}` block is copied directly into the output. allowing the inclusion of header files and additional C code."

Sometimes you want to really call attention to a piece of text. You can center it in the column like this:

`_1008e614_Vector_p`

and people will really notice it.

The noindent at the start of this paragraph makes it clear that it's a continuation of the preceding text, not a new para in its own right.

Now this is an ingenious way to get a forced space. `Real *` and `double *` are equivalent.

Now here is another way to call attention to a line of code, but instead of centering it, we noindent and bold it.

```
size_t :  fread ptr size nobj stream
```

And here we have made an indented para like a definition tag (dt) in HTML. You don't need a surrounding list macro pair.

> `fread` reads from `stream` into the array `ptr` at most `nobj` objects of size `size`. `fread` returns the number of objects read.

This concludes the definitions tag.

7.3 How to Build Your Paper

You have to run `latex` once to prepare your references for munging. Then run `bibtex` to build your bibliography metadata. Then run `latex` twice to ensure all references have been resolved. If your source file is called `usenixTemplate.tex` and your `bibtex` file is called `usenixTemplate.bib`, here's what you do:

```
latex usenixTemplate
bibtex usenixTemplate
latex usenixTemplate
latex usenixTemplate
```

7.4 Last SubSection

Well, it's getting boring isn't it. This is the last subsection before we wrap it up.

8 Acknowledgments

A polite author always includes acknowledgments. Thank everyone, especially those who funded the work.

9 Availability

It's great when this section says that MyWonderfulApp is free software, available via anonymous FTP from

```
ftp.site.dom/pub/myname/Wonderful
```

Also, it's even greater when you can write that information is also available on the Wonderful homepage at

```
http://www.site.dom/~myname/SWIG
```

Now we get serious and fill in those references. Remember you will have to run latex twice on the document in order to resolve those cite tags you met earlier. This is where they get resolved. We've preserved some real ones in addition to the template-speak. After the bibliography you are DONE.

Notes

[1] Remember to use endnotes, not footnotes!