按Firepower服务处理单流大型会话（象流）

目录

简介

本文档介绍为什么单个流不能消耗Cisco Firepower设备的全部额定吞吐量。

背景信息

任何带宽速度测试网站或任何带宽测量工具（例如iperf）的输出结果可能不显示Cisco Firepower设备通告的吞吐量额定值。同样，通过任何传输协议传输超大文件不会显示Firepower设备通告的吞吐量额定值。发生这种情况是因为Firepower服务不使用单个网络流来确定其最大吞吐量。

按Snort处理流量

Firepower服务的底层检测技术是Snort。在Cisco Firepower设备上实施Snort是处理流量的单线程过程。根据流经设备的所有流的总吞吐量，对设备进行特定额定。设备应部署在企业网络上，通常靠近边界边缘，可与数千个连接配合使用。

Firepower服务使用负载均衡来将流量负载均衡到多个不同的Snort进程，其中一个Snort进程在设备上的每个CPU上运行。理想情况下，系统会在所有Snort进程间均衡流量负载。Snort需要能够为下一代防火墙(NGFW)、入侵防御系统(IPS)和高级恶意软件防护(AMP)检测提供适当的情景分析。为了确保Snort最有效，将来自单个流的所有流量负载均衡到一个Snort实例。如果来自单个流的所有流量未均衡到单个Snort实例，则系统会被规避，并且流量会以Snort规则不太可能匹配或文件片段不连续的方式分布，以便AMP检测。因此，负载均衡算法基于可唯一标识给定连接的连接信息。

具有Firepower服务和NGIPS虚拟的ASA中的2元组算法

在具有Firepower服务平台和下一代入侵防御系统(NGIPS)虚拟的自适应安全设备(ASA)上，流量采

用2元组算法进行负载均衡，以便Snort。此算法的数据点为：

- 源 IP
- 目的 IP

Firepower和FTD设备上软件版本5.3或更低版本中的3元组算法

在所有以前的版本（5.3或更低版本）上，流量负载均衡到使用三元组算法的Snort。此算法的数据点为：

- 源 IP
- 目的 IP
- IP 协议

具有相同源、目标和IP协议的所有流量均负载均衡到Snort的同一实例。

Firepower和FTD设备上软件版本5.4、6.0及更高版本中的5元组算法

在版本5.4、6.0或更高版本上，使用5元组算法将流量负载均衡到Snort。考虑的数据点包括：

- 源 IP
- 源端口
- 目的 IP
- 目标端口
- IP 协议

向算法添加端口的目的是在流量中有特定源和目标对占很大部分时更加均匀地平衡流量。通过添加端口，高阶临时源端口必须不同于每个流，并且必须更均匀地添加附加熵，以便将流量均衡到不同的snort实例。

总吞吐量

设备的总吞吐量根据所有发挥最大潜能的Snort实例的总吞吐量进行测量。用于测量吞吐量的行业标准实践适用于具有不同对象大小的多个HTTP连接。例如，NSS NGFW测试方法可测量设备的总吞吐量（44k、21k、10k、4.4k和1.7k对象）。由于HTTP连接中涉及的其他数据包，这些数据包会转换为从约1k字节到128字节的平均数据包大小范围。

您可以估计单个Snort实例的性能等级。以设备的额定吞吐量为例，除以运行的Snort实例数。例如，如果设备的IPS平均数据包大小为10 Gbps，且该设备有20个Snort实例，则单个实例的大致最大吞吐量为每个Snort 500 Mbps。不同类型的流量、网络协议、数据包大小以及整体安全策略的差异都可能影响设备观察到的吞吐量。

第三方工具测试结果

使用任何速度测试网站或任何带宽测量工具（如iperf）进行测试时，会生成一个大型单流TCP流。这种大型TCP流称为“象流”。象流是单个会话，运行时间相对较长的网络连接消耗大量或不成比例的带宽。此类流分配给一个Snort实例，因此测试结果显示单个Snort实例的吞吐量，而不是设备的聚合吞吐量额定值。

观察到的症状

观察到高CPU

大象流的另一个明显效果是snort实例高cpu。这可以通过“show asp inspect-dp snort”或外壳“top”工具来查看。

```
> show asp inspect-dp snort

SNORT Inspect Instance Status Info

Id  Pid   Cpu-Usage  Conns   Segs/Pkts  Status tot (usr | sys)
--  ----  ---------  -----   --------   ----------

0  48500  30% ( 28%|  1%)  12.4 K      0        READY
1  48474  24% ( 22%|  1%)  12.4 K      0        READY
2  48475  34% ( 33%|  1%)  12.5 K      1        READY
3  48476  29% ( 28%|  0%)  12.4 K      0        READY
4  48477  32% ( 30%|  1%)  12.5 K      0        READY
5  48478  31% ( 29%|  1%)  12.3 K      0        READY
6  48479  29% ( 27%|  1%)  12.3 K      0        READY
7  48480  23% ( 23%|  0%)  12.2 K      0        READY
8  48501  27% ( 26%|  0%)  12.6 K      1        READY
9  48497  28% ( 27%|  0%)  12.6 K      0        READY
10 48482  28% ( 27%|  1%)  12.3 K      0        READY
11 48481  31% ( 30%|  1%)  12.5 K      0        READY
12 48483  36% ( 36%|  1%)  12.6 K      0        READY
13 48484  30% ( 29%|  1%)  12.4 K      0        READY
14 48485  33% ( 31%|  1%)  12.6 K      0        READY
15 48486  38% ( 37%|  0%)  12.4 K      0        READY
16 48487  31% ( 30%|  1%)  12.4 K      1        READY
17 48488  37% ( 35%|  1%)  12.7 K      0        READY
18 48489  34% ( 33%|  1%)  12.6 K      0        READY
19 48490  27% ( 26%|  1%)  12.7 K      0        READY
20 48491  24% ( 23%|  0%)  12.6 K      0        READY
21 48492  24% ( 23%|  0%)  12.6 K      0        READY
22 48493  28% ( 27%|  1%)  12.4 K      1        READY
23 48494  27% ( 27%|  0%)  12.2 K      0        READY
24 48495  29% ( 28%|  0%)  12.5 K      0        READY
25 48496  30% ( 30%|  0%)  12.4 K      0        READY
26 48498  29% ( 27%|  1%)  12.6 K      0        READY
27 48517  24% ( 23%|  1%)  12.6 K      0        READY
28 48499  22% ( 21%|  0%)  12.3 K      1        READY
29 48518  31% ( 29%|  1%)  12.4 K      2        READY
30 48502  33% ( 32%|  0%)  12.5 K      0        READY

31 48514 80% ( 80%| 0%) 12.7 K 0 READY <<< CPU 31 is much busier than the rest, and will stay
busy for while with elephant flow.

32 48503  49% ( 48%|  0%)  12.4 K      0        READY
33 48507  27% ( 25%|  1%)  12.5 K      0        READY
34 48513  27% ( 25%|  1%)  12.5 K      0        READY
35 48508  32% ( 31%|  1%)  12.4 K      0        READY
36 48512  31% ( 29%|  1%)  12.4 K      0        READY
```

```
$ top
```

```
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND

69470 root       1 -19 9088m 1.0g  96m R   80  0.4 135:33.51 snort     <<<< one snort very busy,
rest below 50%

69468 root       1 -19 9089m 1.0g  99m R   49  0.4 116:08.69 snort
69467 root       1 -19 9078m 1.0g  97m S   47  0.4 118:30.02 snort
69492 root       1 -19 9118m 1.1g  97m R   47  0.4 116:40.15 snort
69469 root       1 -19 9083m 1.0g  96m S   39  0.4 117:13.27 snort
69459 root       1 -19 9228m 1.2g  97m R   37  0.5 107:13.00 snort
69473 root       1 -19 9087m 1.0g  96m R   37  0.4 108:48.32 snort
69475 root       1 -19 9076m 1.0g  96m R   37  0.4 109:01.31 snort
69488 root       1 -19 9089m 1.0g  97m R   37  0.4 105:41.73 snort
69474 root       1 -19 9123m 1.1g  96m S   35  0.4 107:29.65 snort
69462 root       1 -19 9065m 1.0g  99m R   34  0.4 103:09.42 snort
69484 root       1 -19 9050m 1.0g  96m S   34  0.4 104:15.79 snort
69457 root       1 -19 9067m 1.0g  96m S   32  0.4 104:12.92 snort
69460 root       1 -19 9085m 1.0g  97m R   32  0.4 104:16.34 snort
```

使用上述5元组算法，将始终向同一Snort实例发送长生命流。如果Snort中有大量活动的AVC、IPS、文件等策略，则在Snort实例上CPU的使用率会较高（超过80%），持续一段时间。添加SSL策略将进一步增加CPU使用率，从而降低SSL解密的计算成本。

许多Snort CPU中的少数CPU的CPU使用率较高并不会导致严重警报。NGFW系统在对流执行深度数据包检测时的行为，这可以自然地使用CPU的大部分。作为一般准则，NGFW不处于严重的CPU耗竭状态，直到大多数Snort CPU超过95%且保持95%以上且出现丢包。

以下补救将帮助解决因大象流导致的高CPU情况。

补救

智能应用旁路(IAB)

软件版本6.0引入了称为IAB的新功能。当Firepower设备达到预定义的性能阈值时，IAB功能会查找符合特定标准的流，以便智能地绕过可缓解检测引擎压力的流量。

> **提示：**有关IAB配置的详细信息，请点击。

识别并信任大流量

大流量通常与高使用率低检查值流量有关，例如备份、数据库复制等。这些应用中，许多都无法从检测中受益。为避免出现大流量问题，您可以识别大流量并为它们创建访问控制信任规则。这些规则能够唯一地识别大型流，允许这些流未经检查而通过，并且不受单个Snort实例行为的限制。

> **注意：**要确定信任规则的大流量，请联系Cisco Firepower TAC。

相关信息

- 使用智能应用旁路的访问控制

- 技术支持和文档 - Cisco Systems