UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Faculty of Science and Technology
Department of Computer Science

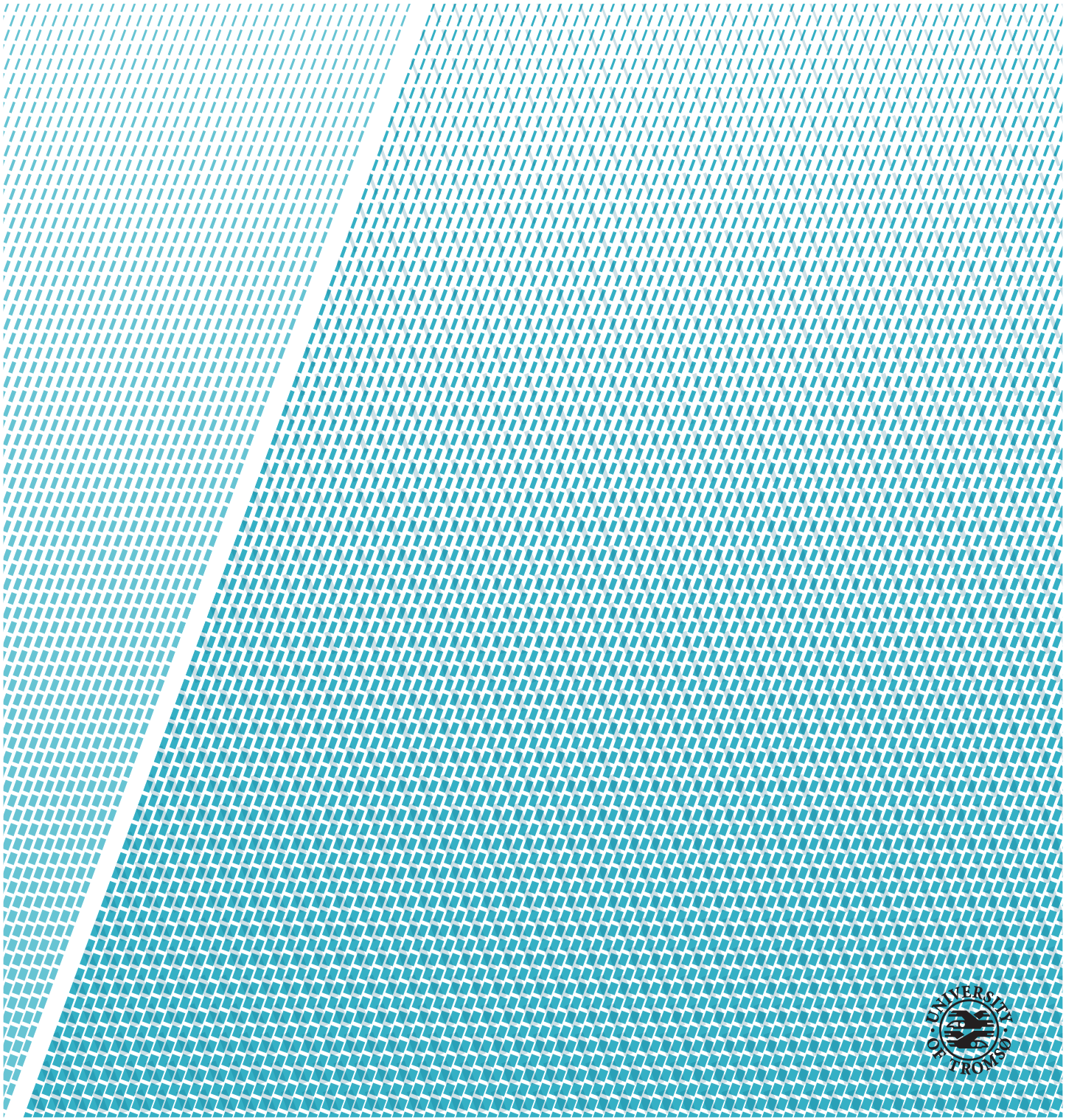
Internet of Things DDoS mitigation

Preventing DDoS attacks using learning algorithms on limited hardware

—

Peter Munch-Ellingsen

INF-3981 Master's Thesis in Computer Science – October 2017



To family and friends.

You are invaluable.

“The more constraints one imposes, the more one frees one’s self. And the arbitrariness of the constraint serves only to obtain precision of execution.”
–Igor Stravinsky

Abstract

DDOS attacks are becoming more and more common, and threatens the current infrastructure of the internet. Cheap new IOT devices have led to a lot of new devices that are poorly secured and can easily be compromised and used for such nefarious purposes. While there are many attempts at solving this problem this thesis looks at a solution which could be applied to typical home router. This would stop malicious traffic before even hitting the internet, as a compliment to the greater effort.

IOT devices typically have fairly simple traffic patterns during normal operations. The system tries to learn these patterns in order to block traffic which would be outside of normal. A home router however is an extremely limited device from a hardware perspective, so a balance has to be struck between learning capability and resource consumption. This becomes especially apparent when considering that most of the chips in home routers doesn't even support floating point operations, which are commonly used for various learning methods.

The proposed system, with the accompanying implementation, shows promising results throughout the testing suite while remaining very low in resource consumption. However dealing with false negatives and implementing the result in a QOS algorithm are still difficult questions. Over all however the solution shows promise and by implementing something like this along with other existing DDOS mitigation efforts a substantial dent can be made in the viability of these attacks.

Acknowledgements

I would like to thank the University of Tromsø for the great study program they offer, which have culminated in this thesis. My supervisor Anders Andersen for his valuable feedback and guidance, and our student counsellor Jan Fuglesteg without whom I would have missed key steps in the process of getting the thesis approved.

I would also like to thank my family and friends, they have provided much comfort and help both throughout the thesis writing and in general. I wouldn't be the same without them.

My father especially with his bluntness about the difficulty of this project and his encouraging words "why did you choose something so hard?", it really set the trouble I'd put myself into in perspective.

A huge thanks is also in order to the Nim community which has been very helpful while learning the language before the thesis and while improving my knowledge of it during the work.

Contents

List of Figures

List of Tables

1

Introduction

Since its inception in the seventies the internet has remained surprisingly unaltered. Our current internet backbone still runs on many of the same systems that were developed by various research teams back when it was only conceptualized to transfer data between large institutions. It has however come a long way since then and it's currently estimated that about 4 billion[1] people today, approximately 45% of the worlds population, have access to the world-wide web from their own home.[1] Most of these users are daily visitors[2] and estimates for data rates range close to 100 Exobytes pr. month[3]. Safe to say the internet is a crucial part of many peoples lives, with many individuals and businesses being dependent on it for their daily activities. It is alarming then that we've started to see more frequent and larger attacks on core infrastructure that has the potential to seriously disrupt the internet as a whole. Back when it was developed this was unlike anything they had ever imagined for their simple information interchange system, and as such the internet is designed more for open-ness than security, for better or for worse. But as the internet has grown older there have of course been many revisions to mitigate some of it's initial downsides.

1. short-scale billion

1.1 Attacking the internet

The internet capacities of even casual users have also grown exemplified by broadband having recently been re-defined by the FCC to mean 25 Mbit/s upload speed and 3 Mbit/s download speed (up from their earlier definition of 4/1). This is good for consumers but it also means that each user have a larger than before potential for havoc. The most common and disruptive attack that we see today is a so called Denial of Service (DOS) or it's worse relative the Distributed Denial of Service (DDOS). These attacks focus on creating so much bogus traffic for a certain service that it is unable to serve it's intended users, thus denying service. Problematically for these services the traffic generated is often masked as legitimate traffic which means that deciding what to throw away and what to actually respond to can be quite difficult. DoS attacks are nothing new, but the rise in DDoS attacks were many malicious units (often without the users knowledge) work together is quite alarming. A recent attack on DNS provider DynDNS has been reported to contain as many as 100,000 malicious nodes generating over a terabit of data per second[4]. While 100,000 nodes might seem much it is actually quite small in botnet terms but shows the incredible attack strength such networks have, bringing down a large chunk of the visible internet.

1.2 The new threat

And along with this large rise in domestic internet speeds and capacity there is also a trend of new devices which are not as closely monitored as devices like PCs and phones which interface directly with humans. While the definition is still vague and quite broad these devices are referred to as Internet of Things (IOT) devices. What often defines such devices are that they are rather autonomous, simple devices delivering a single service through the internet for users. Examples of such devices are home entertainment systems, printers (those connected to the internet), surveillance cameras, weather stations, home automation systems, and even IP phones. The problem with these devices are that they often have very lacking security, being connected to the open internet, and having access to the aforementioned increased bandwidth. This combination means that it's easy for outside assailants to gain access to them and use them for nefarious purposes. The recent Mirai botnet for example gathers it's nodes by simply trying a list of common username/password combinations to log in to devices over things like telnet or SSH. Many of these easy to target devices however have lacking computational power (but this is also increasing) and store little personal information about users which

makes them quite uninteresting for the more traditional kinds of attacks.[2] But for internet disrupting attacks the only thing required is a strong connection and a surprisingly low amount of computational power. IoT devices therefore is a great source of attack power for such malicious intents, and the limited user interaction means that devices can stay infected but undetected for very long periods. The aforementioned Mirai for example even disappears after a simple reboot, but since these kinds of devices are typically kept on they stay infected.

1.3 Amplification and exploits

As previously noted these kinds of attacks can be very hard to distinguish from regular traffic and thus hard to stop. To make matters worse the current IP protocol doesn't enforce actually supplying ones own IP as the sender address. So even if one package is defined as bad there is no simple way of stopping all traffic from that sender. This has been proposed in a 13 year old best practice named BCP 38[5], but adoption has proven to be slow with most of the internet still allowing traffic to originate from any IP. There is also a tried and true method for amplifying a DoS attack (and by extension DDoS attacks) by finding a service that generates a larger response than the request and setting the sender as the node to attack. This will trick the service to replying to the intended target of the attack with a response that is larger than what the initial node had to use therefore amplifying the attack. One typical target of such request are DNS services which can generate responses up to 60-70 times larger than the request. But many other options exists although some of the worst offenders such as the NTP clock server monitoring feature generating over 200 times the request size are being patched away. Amplification is one of the things that would be impossible or at least considerably harder if BCP 38 was implemented across the board.

1.4 Current mitigation efforts

Along with BCP 38 there have been many efforts to not solve the problem on a systems level but rather trying to sort and deal with the traffic. Providers like Incapsulata and Cloudflare are implementing these solutions and providing them as a service to online services. Typically they rely on some sort of

2. For more information about the possible attack vectors of machines under foreign control see Brian Krebs diagram here: https://krebsonsecurity.com/2012/10/the-scrap-value-of-a-hacked-pc-revisited/

fingerprint or traffic pattern recognition to decide which traffic to let pass through to the service and which traffic to stop. However they can only stop as much traffic as they have resources to manage. This means that these mitigators need to be able to handle the multi-terrabit traffic that can be sourced in a DDoS attack. And this remains a constant arms race, with poorly secured IoT devices and increased residential speed increases giving the botnet owners a possible edge.

1.5 Problem statement

This thesis considers the problem of DDOS attacks from a new perspective and is looking to implement a localized system wherein routers with connected IOT devices are able to detect anomalous traffic and stop these devices from participating in a meaningful way to said attacks. This solution is based on the premise that most such devices are not participating with the consent or even knowledge of their principal owners and would not be allowed to take part in malicious activities if those were given the choice.

Since the typical home router is fairly limited in their hardware it is important that the detection is done in such a way that it consumes a minimal amount of resources. These devices are seldom found with more than a couple megabytes of memory and run with weak and less capable processors than are typically found in other machines. This limits the kinds of algorithms that can be used for the detection. Another important factor is to not substantially hinder normal traffic, especially if the device is not performing malicious actions. Traffic should be able to flow unhindered through the system.

1.6 Limitations

This thesis focuses on IOT devices. Partially because they have recently been the main target of recruitment for DDOS botnets but also because they typically have a much more predictable traffic pattern. No attempt have been done to implement this for more organic traffic.

After having determined when traffic is considered bad or not it should be stopped or otherwise limited by the router. This is not considered in this thesis as it would heavily rely on how the existing Quality of Service (QOS) algorithms on the coordinating device worked.

In order to be able to perform controlled repeatable experiments it was impor-

tant to be able to control the perceived network traffic. Therefore the current implementation is only written in such a way that it can consume captured network traffic and not implemented on actual hardware. Part of this is also linked with the above reason that it would have to be integrated with other systems.

2

Background and related work

2.1 Denial of Service

Denial of Service attacks have been around for a long while, and since their inception around 1997 have grown to become one of the biggest threats online today. While they typically don't entail any loss of control over services, any leak of sensitive data such as passwords, or destruction of data they still offer the possibility for extortion, removing competition, or simply vandalism. Part of the problem with such attacks is that they don't target any specific vulnerability making them hard to defend against. In fact most devices directly connected to the internet are vulnerable to such attacks just by virtue of being connected. The idea behind a DOS attack is that the malicious traffic generated should make well-behaving accessors drop their connections and therefore not be served by the server. There are many ways of doing this, one way is to use some kind of vulnerability with the server to exhaust it's resources. Maybe the most infamous of such attacks is the so called Slowloris. The idea is simple, open many requests spawning threads on the server, these requests will exhaust the thread-pool of the server. All the connections are then kept alive by slowly sending data over them, imitating a user on a slow network. This uses very little resources on the attackers side, but exhausts all the resources the server has to serve actual users. Another method of attack is simply to flood the target with legitimate traffic, this means that the server is serving actual requests,

but the sender of these attacks simply don't care for the answer but limits legitimate access.

2.1.1 Amplification

As previously mentioned, some flaws with how internet traffic is handled today which makes the attackers able to amplify their attacks. This amplification works by finding endpoints which responds with more and/or larger packets than the initial request and then spoofing the sending address of requests such that this response is not sent to whomever made the request but rather towards the target of the attack. One of the most notorious such amplification schemes uses NTP servers which exposes a debug command to get the last 600 time requests. These servers typically have strong connections as they serve an important role in keeping time synchronized between machines. In fact this kind of amplification represented 85% of DDOS attacks over 100 Gbps back in 2014[6]. However since this kind of amplification depends on a particular fault they are often patched within a fairly short period of time, and after just a couple of months the amount of vulnerable servers were down 90%. These attacks were made even easier by being able to be triggered over UDP which doesn't require a handshake. TCP on the other hand has a three-way handshake which should make it more robust against such attacks but that does not seem to be the case. During a scan of randomly selected IP addresses, also this in 2014, researchers was able to find TCP nodes that would amplify traffic by up to 80,000x of the request size[7]. Some of these amplifications (although quite a bit smaller), were even during the TCP handshake itself.

2.1.2 Distribution

Even while using amplification DOS attacks still have to rival the bandwidth and capacity of their target server. And by using for example the large NTP servers for amplification it isn't hard to simply disallow traffic from such a service should it be found to misbehave. So in order to be able to compete against the large capacity of typical consumer-facing servers another approach can be used. Instead of having a single attack node multiple attackers may band together and attack simultaneously. This is something which might even happen if a small server suddenly gains a lot of attention. The online community Reddit is a perfect example of such accidental DDOS attacks as users are able to share arbitrary links which may suddenly gather a lot of attention. It even happens so often that the term "Reddit hug of death" has become widespread on the site, referring to showing so much positive attention to a site that it goes down. However gathering enough people with malicious intents to bring down for example a DNS provider would be rather difficult, and organising

the entire thing would be a feat in itself. The alternative is to gain access to unsuspecting users systems and use those for these nefarious purposes. This is typically achieved through a number of various exploits and vulnerabilities of varying sophistication. A target ripe for the picking in this regard is, as previously discussed IOT devices.

2.1.3 Internet of Things devices

While the term IOT is ambiguous and can refer to a few different concepts what is meant by it in this context are devices which are connected to the internet but have no human interaction controlling their network traffic. They might be set up by a human to perform certain internet-related tasks, and might be accessible over the internet by humans, but are in most regards not dependent on human interaction to perform their function. There are several things that makes these devices interesting to potential attackers. Since they don't require human interaction it also means that they are typically not very supervised by humans. This makes a potential malicious take-over harder to detect as it doesn't necessarily interfere with it's operation. IOT devices are also seldom properly secured properly with many still using regular HTTP and leaking credentials over unsecured connections[8]. The recent Mirai botnet also grew rapidly and became quite large simply by testing a list of default password/username combinations, leading to an attack which could easily have been avoided had the devices been properly secured.

2.2 Related work

Trying to stop DDOS attacks is nothing new and there exists many proposed solutions to this problem. Saied, Overill, and Radizk [9] show a proof-of-concept design for an artificial neural network to detect DDOS traffic and cite other papers showing similar solutions. Both CloudFlare [10] and Incapsula [11] also have protection products which can be employed to stop a DDOS attack. Those solutions however only offer taking the bulk of the requests and doing away with them, something which is more about preventing the effect of a DDOS and not stopping the attack in itself. Companies like BullGuard also offer more consumer-facing products to amongst other things protect the less secure IOT devices on your network from being hijacked in the first place[12].

3

Architecture and Design

The solution proposed in this thesis works a bit different than many typical DDOS mitigation efforts. Instead of having the protection on the server side or throughout the network it focuses on local prevention. Most IOT devices participating today in DDOS attacks are not owned and operated by malicious hosts, but rather hijacked by outside attackers. This opens the possibility for setting up a system in which these devices would be kept under strict supervision to determine if it's misbehaving. To be able to separate devices this thesis assumes the software is running in a scenario where it is uniquely able to identify devices, for example as part of a home router. This would allow it to see all traffic by a certain device, and have full control over it's appearance online. Most people however don't need their routers to do much and the hardware in such devices are often pretty limited, this poses an additional challenge and limits what kind of solutions are possible. It is also important to interfere minimally with regular traffic, and since this would be placed with people who might be less technologically inclined operation should also be kept simple, or better yet not require any human interaction. How such a system could work is what is underlined in this section.

3.1 Hardware

Before going into how this system would work it is important to evaluate the target platform. Most home routers today run on fairly modest hardware.

When looking at the most popular custom router firmware DD-WRT they classify routers into categories mostly based on flash storage, their biggest version "mega" only requiring 8MB. The devices typically have limited amounts of Random Access Memory (RAM) as well ranging from 2-64MB, and processors running at clock speeds of about 125-400 MHz. These processors also lack hardware floating point capabilities which means that all floating point calculations would need to be done in software for a punishing performance penalty. Most learning algorithms today use floating point mathematics and would therefore suffer a severe diminishing in performance when running on this hardware.

The small amount of RAM means that each connected device should store as little information as possible, while the processor clock speed limits the amount of operations which would be acceptable to use for the learning. The lack of hardware floating point also means that for an efficient implementation the design would benefit from limiting the use to only integer and bit-wise operations.

3.2 Badness score

Each managed device is assigned a badness score, this is a value that says something about how much recent traffic has deviated from normal operation. By applying this value in QOS algorithms the traffic a certain device can use can be scaled depending on how well it behaves. When the device is not misbehaving this value should be sufficiently low to allow all it's traffic to flow normally. When misbehaving it should increase and throttle the device, potentially even cutting it off entirely. As mentioned in the limitations section this implementation does not include any QOS algorithms and as such doesn't make any attempt at controlling the traffic based on the badness score. However for a discussion on how the badness score should be used in such algorithms there is mention of this in the discussion section.

There are many different approaches that could be taken to determine this score, and some of the simplest might be to have a set of known bad traffic patterns and match the current traffic against those. This would be similar to how anti-virus has a list of fingerprints for known viruses that it then scans the drive for. Such a system would rely on both keeping metrics for each device and a good database of patterns to match against. Matching many patterns are also problematic as it would require a lot of processing power. A bot could also be programmed to avoid the most common patterns, something which might limit its usability but would still allow it to defeat the check, this is especially true if the patterns had to be very narrow to allow regular traffic through. So

short of simply rate-limiting anything that isn't a human-operated device it would be hard to make such a system robust enough.

The solution proposed here is to create a fairly basic but functional learning algorithm that doesn't keep logs of traffic history, but rather keeps aggregated values for what kind of traffic is coming through. This system would tune a set of parameters while data flowed through from the monitored device and if the observed traffic went outside of these parameters flag the traffic as bad.

3.3 Testing methodology

As a proof of concept this system takes in capture files in the `libpcap` format created by `tcpdump`. This is a very simple packet capture format which closely mimics how packages are actually transferred on the wire and offers the minimum of information that should always be available for the router. This approach was chosen as it would make running the tests much easier instead of using live data from infected devices. In order to test the system a couple different traffic patterns were established as a baseline that could be expected for typical IOT devices.

Repeated request , a simple `HEAD` request sent to a server and an answer received at a set interval. This is to mimic a simple system that regularly fetches data from a server for display purposes or to remotely control the device.

Streaming , a Real-Time Streaming Protocol (RTSP) stream continuously running. This is to mimic the common CCTV camera streaming it's data.

Upload , Secure File Transfer Protocol (SFTP) upload excerpt of a single large file. Not a particularly common case for IOT devices, but similar in traffic amount to a DOS attack.

DDoS , for testing purposes a Transmission Control Protocol (TCP) SYN flood was used. This is a pretty standard DOS attack that exploits the TCP three-way handshake.

These patterns were captured using the readily available network protocol analyser Wireshark[1]. The traffic was filtered to a specific IP as not to contaminate it with background traffic from the capturing system. To allow capture files taken on different days, and to mimic different IP addresses of the sender/receiver,

1. https://www.wireshark.org/

the test running script incorporates a mechanism to stitch together and change captures. They are stitched together with one continuous string of timestamps based on their original time difference, and the IP of both the sender and receiver can be changed to mimic sending to or receiving from a different host than the capture was made against. This was done in a minimally obstructive way to keep the captures as true to their original state as possible.

3.4 Detecting change

As previously mentioned the system tries to determine a badness score for each device based on what normal traffic for a given device consists of. Most IOT devices have fairly simple and repeatable traffic patterns which makes this easier. Since the hardware is limited, using neural networks or similar systems would likely not be possible without severely limiting the amount of devices it could keep track of. Therefore a simpler approach is taken, focusing on keeping both memory and the amount of operations (especially floating point operations) at a minimum. To achieve this a couple of relevant metrics was chosen. The three things currently considered by this design are:

IP Address , to detect traffic going to never before visited hosts

Packet size , the normal traffic is not unlikely to have different package sizes to an attack

Time between packets , change in velocity of packages is a good indicator of a DOS attack

The latter two are continuous values which means that they are on a range and can be compared by value, and not just for equality. IPs on the other hand are entirely discrete and comparing one number to another doesn't make much sense.

3.4.1 Change in continuous values

Multiple approaches was tested to get good results for continuous values. Originally the system was based on a min/max approach in which the minimum and maximum values observed were compared to the current packet. The minimum and maximum would tend towards each other so that the window was always kept at it's minimal size. This worked fine for the time between packets, but was giving lots of false negatives when used on the packet size. The issue turned out to be that when fed with very small packets (such as those

in HEAD requests and SYN floods) the amount to shrink by would tend to be lower than zero, which was a problem without floating point mathematics. If it was rounded to 0 there would be no change, and therefore the window could stay too big. If it was rounded to 1 the change would be too large and cause the window to collapse over valid data.

Changing floats for probability

Since the system is not really dependent on having the exact sub-integer precision of its state values but rather on how slow or fast their progression were a simple replacement was found. The solution to this was to use probabilities, if for example we wanted to subtract 1/10th from a number we can subtract one with a probability of 1/10. The probability to add exactly one over ten iterations is not very high, but the probability to have added on average one over many iterations of ten approaches 100%. There are many high-quality random number generators out there, many of which don't require many CPU cycles to generate pseudo-random numbers. However the current implementation uses a slightly different approach. Instead of generating random numbers a couple of bits are taken from the current system time in it's highest resolution. These bits are AND-ed with an appropriate mask and the result is checked against a target result, when it fulfils the target the value is incremented. As long as the possible range of decimal values aren't too great this is a sufficient amount of randomness.

3.4.2 Change in discrete values

Discrete values offer a different challenge to continuous ones. The problem here is that comparing minimum and maximums doesn't make much sense. For example comparing the IP address `129.242.219.53` to a very numerically similar IP `128.243.217.54` would yield a low difference, but is just as dissimilar as any other IP address with all four fields changed. So in order to create a difference that makes sense an IP mask system is used. The mask is created by bitwise operations applied to pseudo-random bits which slowly, by applying the same IP address, turns into a 1:1 mask for that IP. If more IP addresses are applied over each other the mask will be an amalgamation of the addresses, but yielding consistent differences to each of the IPs. These differences however are continuous and can be passed through the same system as used for packed sizes or the time between packages. The mask also deteriorates slightly with each new package, such that any IP address that hasn't been connected for a long while will be removed from the mask. This system allows the device to connect to many different IP addresses without having to store a large list of them and checking through the list. If too many IP addresses are used then all

traffic will simply be marked as good by this test.

3.5 Determining device category

All of this depends on the devices surveilled to have rather predictable traffic patterns. More complex learning algorithms could potentially be used to handle organic traffic but with such restricting hardware it's probably not feasible to implement without heavy performance penalties. And seeing how most of the traffic in DDOS attacks originate from these devices it would likely make a strong impact on their capabilities if widely implemented. But in addition to being able to actually supervise the devices a system to detect which devices to supervise also needs to be put in place, lest devices with organic traffic gets a badness score which assumes non-organic traffic. Since this implementation only deals with analysing the traffic this is not implemented in the code related to this thesis. However one system which might work is to implement a Captive Portal (CP) which redirects all traffic to a site which then requires some user input to accept the device as an organic one. This should be sufficiently complicated that a simple bot should not be able to pass the test. If no response has been heard from the CP endpoint the mitigator can assume the device is a simple IOT device and add it to it's list of devices to watch. Logging in through the router configuration portal should also allow devices to be changed from one state to another. This might not be the most sophisticated of methods but should work decently well to classify devices. And as most users have seen these when using public hotspots they shouldn't offer too big a challenge for the less technologically inclined users.

3.6 Learning badness

Since this is a system which tries to learn traffic an important question arises. When should new traffic be added to our measure of good traffic and when should it be blocked from accessing the network? And along the same lines how can we make sure that a possible attacker is not slowly re-learning our measure for good and bad traffic to circumvent the penalties or trick the system into penalizing good traffic. This is something which comes tightly coupled with how the system would be integrated with existing QOS algorithms, and therefore out of the scope, but it is discussed in more detail later in the discussion chapter.

4

Implementation

As previously mentioned the system was implemented to run on a regular workstation to aid in testing. The implementation is however written in such a way that all the learning and checking is in its own separate module so that it could be reused in different applications. An example of such applications are the various test tools that was used during development, most notably the program capable of stitching together various capture files and playing them through the system. To enable code-reuse for an eventual implementation on actual hardware the program was written in Nim[1] which compiles down to C and can be used to target most, if not all, hardware C can run on.

4.1 Random numbers and float simulation

As mentioned in the previous chapter dealing with growing or shrinking a number with a value smaller than one is important to the system. It was mentioned how this is achieved through using random numbers. The current implementation uses this when dividing a number by eight (implemented as a right shift by three operation). This means that we have three bits of information which gets discarded, so if our number is only in those three bits the number becomes zero. There are only eight possible numbers that fulfil this condition, and they are hard-coded into the implementation. By picking up to

1. https://nim-lang.org/

Num.	Prob.	Mask	Check
0	0	0	
1	1/8	0b111	==
2	1/4	0b11	==
3	3/8	0b11 and 0b111	==
4	1/2	0b1	==
5	5/8	0b111 and 0b1	==
6	3/4	0b11	>0
7	7/8	0b111	>0

Table 4.1: Table showing how numbers and their probabilities match the bit-masks

three bits of "random" data from the end of the milli- or nanosecond resolution clock and checking their value it is possible to achieve the probabilities we need. This is illustrated in table 4.1. The first column contains all numbers which would be truncated to zero by the right shift operation. The second column contains the probability of adding a one for each number. In the third column we find the mask or masks that are AND-ed with the time, and the fourth column lists the operation to compare each result. Those marked with == imply that the result should be equal to the mask, and those with >0 are checked if they are higher than 0. Checking if the result is equal to the mask is the same as checking that all the bits are set, and checking if it is greater than zero is the same as if any bit was set. If we assume that the lower three bits of the clock is random it is easy to see how for example the compound probability $P(\text{bit 0 is set}) \times P(\text{bit 1 is set}) \times P(\text{bit 2 is set}) = 1/2 \times 1/2 \times 1/2 = 1/8$ is the same as checking if the AND-ed mask is the same as the original (all bits have to be set). Similarly looking at the the mask for the probability $7/8$ we can see that checking that one of the eight outcomes does not happen will generate the wanted probability. This of course assumes that the last bits of the nanosecond time-stamp are random, and that a possible attacker is not able to construct packages that were processed at a specific time.

4.2 Continuous values tracking

There are two different kinds of continuous values tracked by this system, packet size and distance between packets. Distance between packets is only tracked when the connection is continuously transmitting, implemented here as receiving packets within 50 times the largest packet gap. This is to differentiate between the velocity of an actual transmission and breaks in transmission altogether. In the HEAD request test for example a HEAD request is comprised of multiple packets being sent with varying distance between them, and long

breaks of inactivity of 30 seconds were no packets are sent. This would lead to an overall average which would be very lenient in what it would accept.

4.2.1 Minimum/Maximum tracking

This is the first approach which was implemented, and is currently in use for one of the two continuous values. It tries to keep track of the highest and lowest value the system has seen thus far. Of course this would be wasted if a single huge or tiny value was received, so for each packet it also tries to shrink the series of accepted variables. However only shrinking the series would lead to a lot of values ending up outside the range. For example if the system sent three packets followed by one short the range would shrink from the bottom while the large packets were received only to always invalidate the small packet. This is solved by storing the amount of shrinkage that has taken place since the last value that stretched the bound in each direction. When a packet arrives that is out-of-bounds it is checked against this value and is not flagged as bad if it is within the limit. During testing this system was the one that delivered the most stable performance for the packet distance metric and is were it is still used for now.

4.2.2 Distance tracking

While the minimum/maximum tracking worked fairly well for the packet distance it was giving poor results for the packet size as will be shown in the following testing chapter. To improve the results a second tracking method was developed which instead of tracking the upper and lower bounds tracks just an average and a deviation. This not only means less memory overhead for each device, but also proved to give more stable results for both the packet size and the IP deviation metrics. The tracking here works by increasing an average value in the direction of the current packet size as well as shrinking or growing a difference value thus creating a range around the average. Whether to grow or shrink the difference is based on how close the packet size is to the average. Anything less than half the difference away will shrink the difference, and conversely anything that's over half the difference will increase the difference. The amount to change by is the aforementioned 8th of the distance which led to the float simulation described above. Whenever the packet size hits completely outside the this number is further divided by 2 without any float simulation for a small change of 1/16th of the difference.

4.3 Discrete value tracking

Discrete values offer a quite different challenge from continuous value when it comes to tracking. The tracking here is based on comparing the difference between one value and another based on their bit patterns instead of their actual value. This means that the only data stored is a mask of the values the same size as the value itself. So for an IP address this would be 32 bits as each value in the IP is 8 bits each. However the difference of the comparison doesn't really suffice as it can be normally noise if more than one IP is present in the sample. This is solved by passing the difference of the IP check through one of the continuous value tracking methods. By doing this the system is able to distinguish between a normally noisy sample and an actual badness.

In order to track a value the system sets one pseudo random bit from the input value in the current mask. This pseudo random value can, as described earlier, be gathered from any applicable source but in the current implementation it is bits taken from the current time. After the mask is changed the old value is compared with the new value and the difference (in bits) is returned. This is the value that is passed through the continuous value check.

4.4 When to change

During the various iterations of this implementation the system would often stagnate and freeze in a certain state. This was due to changes not being applied when a bad value was detected, which became especially apparent when the system was completely fresh and initialized to very unrealistic values. The current implementation therefore tries to always make *some* change to the internal state even for bad packets to ensure that such freezes won't happen. This means that given enough time the system will drift out of a potential freeze and into a usable state.

5 Testing

As mentioned in the architecture and implementation chapters one of the metrics to decide if traffic is bad or not is the change in IP addresses. The system also checks only outbound traffic at the moment. This combination meant that a small utility to change the IP addresses in a Packet CAPture (pcap) file had to be written. To make the captures more pure and only contain the data they were supposed to a simple utility to select only a subset of packets were written. These two utilities were used to create four captures for the four scenarios described in the architecture chapter. The third and final utility was one to run through multiple files, stitch them together back to back, and play them through the system while generating output to be passed through `gnuplot` to generate graphs.

5.1 Classification testing

These graphs were then used to gauge how well the system functioned. Graph 5.1 is a typical example of what the system looks like while running.

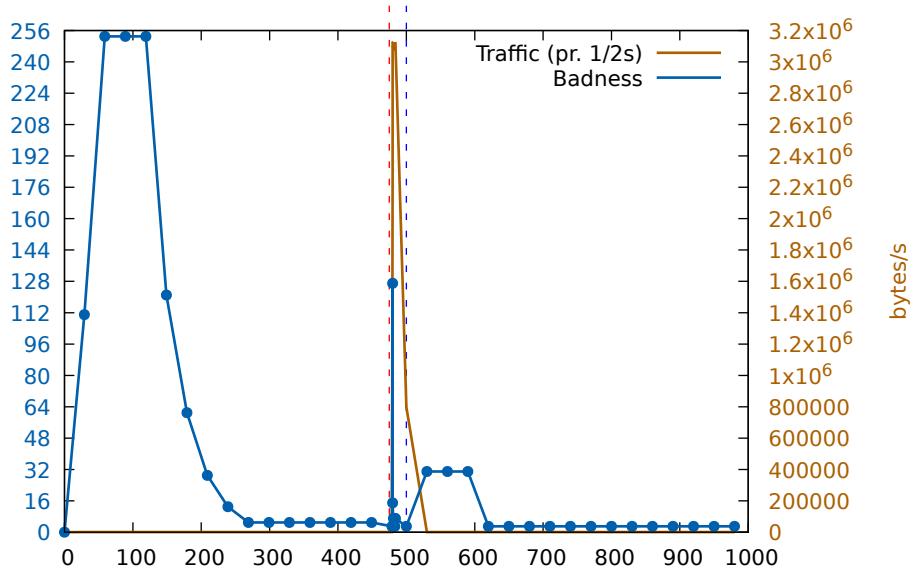


Figure 5.1: Graph showing three sections, HEAD requests, a SYN flood, and more HEAD requests

The start of the graph shows how the system goes from an uninitialized state to being tuned to the data going through it. The badness is very high for the first couple seconds before it eases out and the SYN flood starts at the red dashed line. Note the blue points on the line, the system only performs calculations when sending data so during the 30 second pauses between HEAD requests there is no data. At the red vertical line the SYN flood starts and immediately the cumulative amount of data is seen shooting to the top. During this period the system is pushing a lot of packets in a short timespan but this is only represented as a short spike, in reality there are over half a million packets constituting that spike. We can also see the badness reacting to this change in data, also represented as a single spike before it learns that the SYN flood data is normal. As the calculations are done on each packet it means that the system quickly returns to a state where it considers traffic to be normal, the badness score shown is the highest badness detected during the resolution of the graph of 1/2 of a second. The blue dashed line shows the end of the SYN flood and as the amount of traffic plummets the badness score once again rises. This time it is less pronounced, note how this continues longer that the initial spike as there are now fewer packets per half second and the badness stays high for a full minute.

In the graph shown in 5.2 we can see the opposite composition from the above. Instead of a period of HEAD requests followed by a SYN flood this run is first tuned on a SYN flood, then subjected to a series of HEAD requests. As can

be seen in the graph the result is pretty much as expected given the above results.

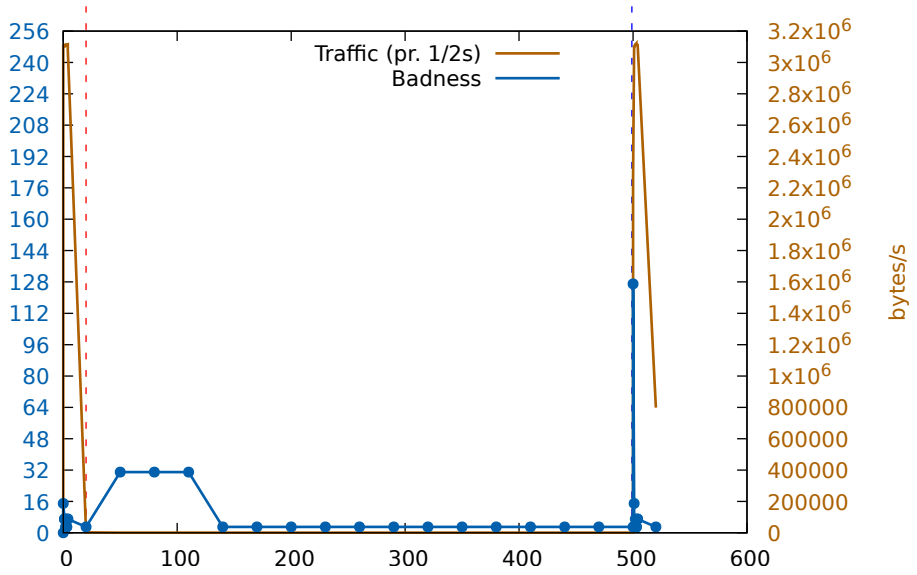


Figure 5.2: Graph showing three sections, a SYN flood, HEAD requests, and another SYN flood

The next graph seen in 5.3 shows much the same as 5.1, albeit on a much shorter time-scale and with much more traffic in the regular sample. This is based on the SFTP upload, followed by a SYN flood and then back to the SFTP upload. The badness is initially high as with the first example, but quickly drops once enough packets have passed the system. Then after the SYN flood starts at the red dashed line the badness spikes before recognising the flood as normal traffic. Later when the SYN flood is over the SFTP data is detected as bad before it to is seen as normal again.

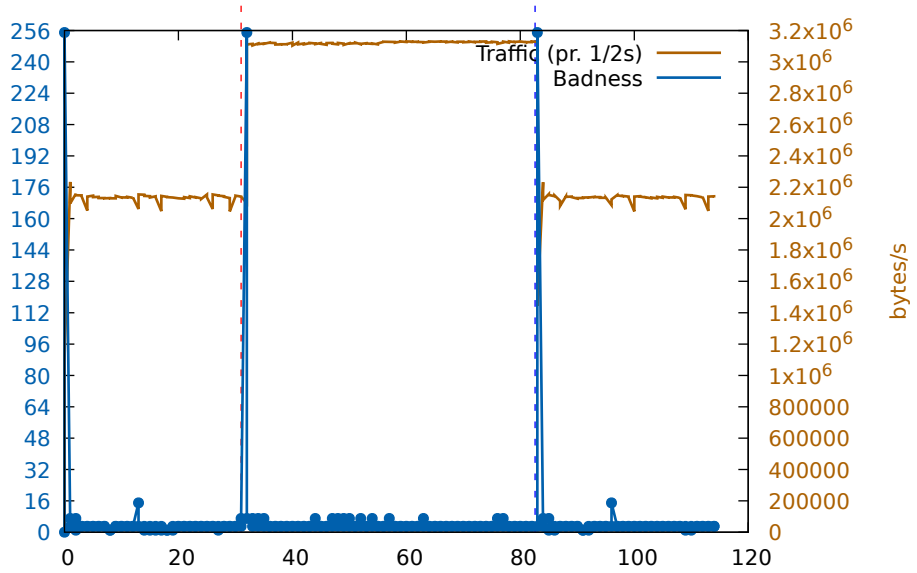


Figure 5.3: Graph showing three sections, SFTP data, a SYN flood, and more SFTP data

As before the inversion is seen in 5.4 where the SYN flood is followed by the SFTP data before going back to a flood.

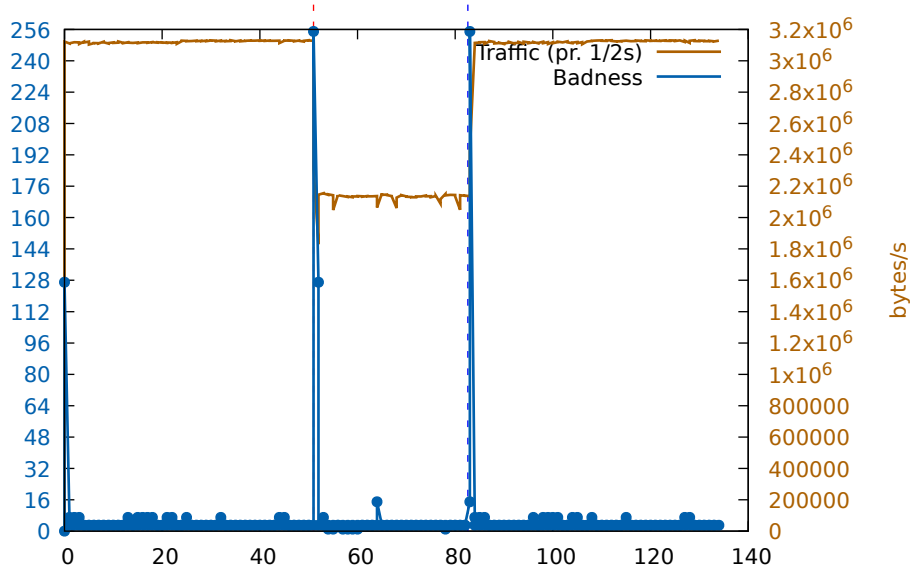


Figure 5.4: Graph showing three sections, a SYN flood, SFTP data, and another SYN flood

Finally the graph in 5.5 shows some different results. Here the system is not able to tune in to the RTSP data, and there are frequent disturbances in the badness score. When the SYN flood starts and stops there is a spike which corresponds to the change in traffic, but with the overall noisy result it would be hard, if not impossible to use this data for anything of value.

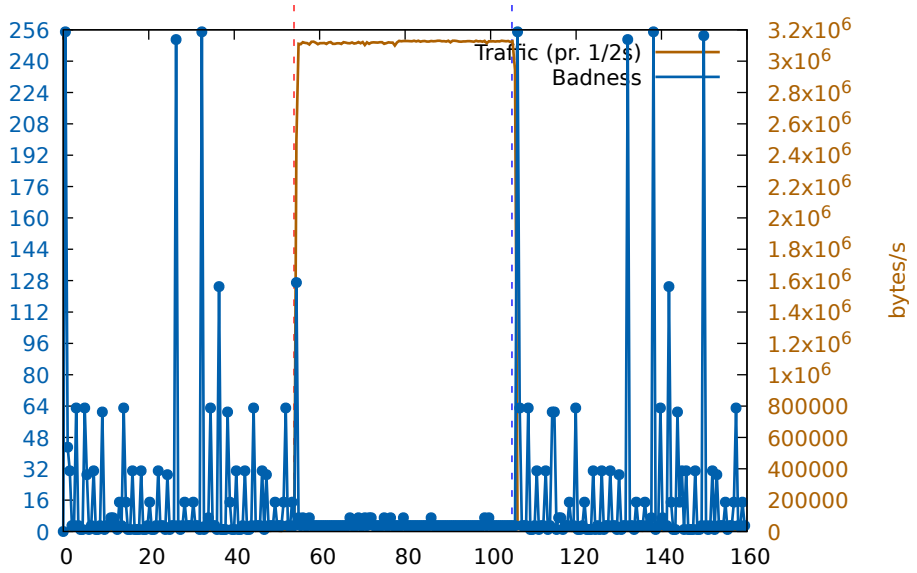


Figure 5.5: Graph showing three sections, a SYN flood, RTSP data, and another SYN flood

5.2 System inspection

As mentioned the target hardware for this system are simple home routers or their equivalents. This means a quite severe limit on what is possible to use in terms of resources. In the current implementation the two trackers are the distance tracker, the minimum/maximum tracker, and the IP tracker. The table in 5.1 shows the memory required for each of the data structures associated with these trackers. The system uses one distance tracker for the packet size, a minimum/maximum tracker for distance between packets, and one IP mask tracker. The IP mask tracker consists of one IP mask and a distance tracker. This means that the entire tracking system uses $distance * 2 + minmax + ipmask = 8 * 2 + 16 + 4 = 36$ bytes of memory per device. This low amount not only means that a lot of devices could be connected at the same time, but also that swapping the memory out of storage would incur less of a performance hit.

Tracker	Memory (bytes)
Distance	8
Min/max	16
IP (mask only)	4
IP w/tracker	12

Table 5.1: Table showing the various trackers and their memory requirements

In terms of CPU operations it's hard to calculate without having an actual hardware implementation. However the typical distance check and update doesn't use much more than 5-10 arithmetic operations (including bit shifts) and a similar amount of branching statements. Again the minimum/maximum solution is a bit more involved, but not considerably so. Considering that a good software floating point implementation is about 10-30 times more cycles than integer arithmetic[13] this is a very small amount of operations compared to more traditional learning algorithms. Of course depending on the performance of the underlying CPU branch predictor it might be beneficiary to implement some of the branching logic with arithmetic should the compiler fail to do so.

6 Discussion

In the previous sections many graphs the system shows that it is able to detect changes in traffic, but there are a few shortcomings. This chapter will go into more details on how the data from the system could be used and what limits it would have from this perspective. There are two main issues with the current implementation, first the badness only spikes on the edge of the data, and secondly it is not always able to fully tune into a data stream and see it as normal data.

6.1 Tuning issues

Throughout this project the system would often fail to properly tune to a set of data. This is still apparent with the RTSP example from above. While it looks fairly bad in the graph there is actually only about 4% of the data which is marked with a badness level of over 20, and no more than about 2% which is above 32 in badness. If this was fed directly into a QOS algorithm however it might, at least for other protocols than RTSP, end up prompting a lot of retransmits and other things which could further deteriorate the badness score.

Since this has been such a prevailing problem throughout the development multiple methods of dealing with these scenarios have been considered, however none have been implemented due to being out of scope. One method

would be to use one of the continuous value trackers to track the badness value itself. This would mean that the system would be able to have some normal badness and still be able to detect a change in the data stream. As seen in the graph in 5.5 the badness drops when the SYN flood starts, which could be considered not-normal for this sample. This system could also be used to simply decide to not track certain devices if their badness fluctuated too much for the system to be able to get a proper reading. If this worked properly it is also conceivable that the system could be applied to more organic data as well, and simply discard devices it didn't understand.

The RTSP example is however especially unfortunate as these devices are quite numerous in the IOT device counts. Upon analysing the data for this example it became apparent that the reason why it is so unstable is simply because it periodically sends packets which are two to three times larger than it's normal packets. It is these packets that triggers the packet size metric and in fact when disabling that part of the system the tracking of this particular case is substantially better as seen in 6.1. This is however not a universal solution as some of the other cases don't give nearly as good results without it. An interesting approach would of course be to include more metrics, and possibly discard metrics that introduce too much noise on the fly. However this is still further outside the scope.

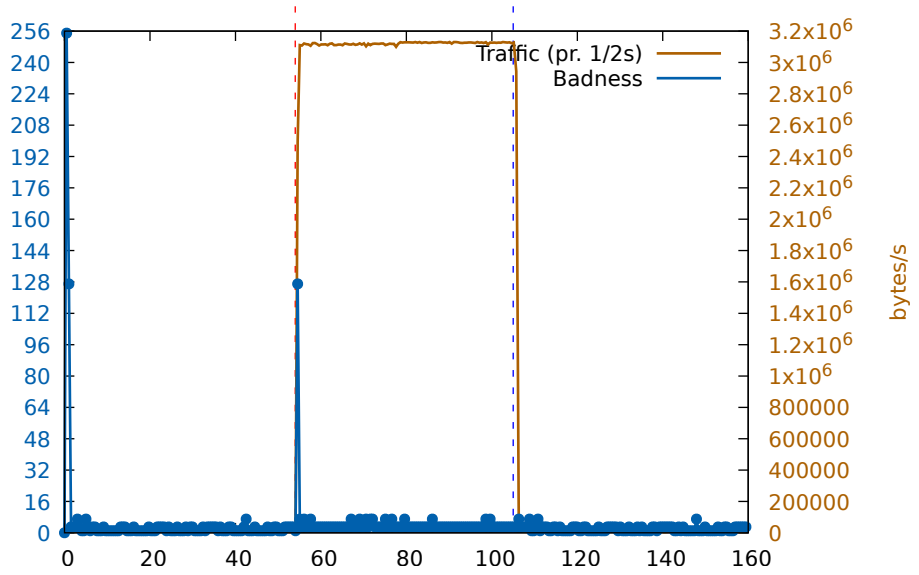


Figure 6.1: Graph showing three sections, a SYN flood, RTSP data, and another SYN flood. With packet size tracking turned off

Yet another solution could very well be to simply increase the amount of packages needed to change a given metric. This is something which would allow the RTSP traffic to include the larger numbers in it's package size metric.

The current implementation has a very short time-scale it operates on. Part of the problem is that the current solution is only based on a per. packet analysis. This means that something like the HEAD request example which only sends about ten packets every 30 seconds would take hours to tune the metrics to it's values. During a real-life operational situation however it very likely that this would be beneficial for the system, but during the initial design phases were various metrics where in consideration and everything was being manually analysed on a per packet level this simply wouldn't have been feasible to implement. Now that the system is more mature and stable however it is certainly an interesting vein of further research possibilities. Doing this would also serve to let bad data me marked as bad for longer.

6.2 Retaining badness

Slightly tied together with the previous issue is the fact that the badness score is very quickly getting back to normal levels. This is again because there is such a high volume of packets passing through the system and each packet has a quite meaningful impact on the metrics tracking information. While initially something which was born from necessity from not having floating point numbers that could deal with very small changes it should certainly be possible to implement a more generalized version of the floating point emulation to overcome this limitation.

Another option, which could also be used in conjunction with decreased sensitivity, is to keep snapshots of the tracking data. For example if the system were to each day at a certain time incorporate the current tracking data in a secondary tracking structure which would remain untouched for the rest of the day. This would mean that the secondary tracking structure would be frozen for the entire day and when a packet with sufficient badness is detected it could switch to use this frozen copy until the data is once again back to being considered good. Doing this would allow the short spikes seen in the graph above to trigger the usage of a filter corresponding to what caused that spike in the first place and thus remain flagging packets as bad. Care must be taken however to not end up in a state of flagging every packet is bad and never returning as has been previously mentioned. This was an issue during development and it causes the system to never be able to return to a normal state as it can't change what is determined as good. Of course if a tiny portion of the bad traffic were to count towards the frozen tracking datas daily change then it would be able to slide out of it, albeit over a very long period of time.

However both of these methods only delays the problem, at some time the system would start considering bad data as normal data, strictly from the

nature of being a system that learns. The first system would probably start doing this after a couple of minutes or hours, depending on how insensitive it actually was. And the second might delay the fact for multiple days, at least if combined with the first solution. But if the utility of botnets are taken into consideration it certainly limits them. Getting an attack up to speed under a system which implemented either, or both, of these mechanisms could possibly take too much time for it to be of any value. The target of the attack, or intermediary infrastructure, would have time to detect and avert the attack. Or if the attack was as a response to a certain event it would be completely out of context if it was carried out days after the fact. These networks are also commercialized, selling DOS capabilities by it's throughput. Having to pay for a ramping up period where the botnet had to be carefully orchestrated to slowly alter it's own traffic pattern without setting of the badness detection would certainly make them less viable. Not to mention that this would be a technological challenge for the writers of botnet code.

6.3 Implementation into QOS algorithms

Most routers today come with some kind of QOS system. These work by giving priority to certain types of data or devices in order to have data that's considered more important pass through the router unimpeded. A typical example would be to prioritize streaming services and Voice over IP (VOIP) to avoid stuttering while other machines in the network are performing routine updates. The system described in this thesis could be implemented as a part of such QOS algorithms such that IOT devices currently experiencing high badness scores would be allotted less bandwidth.

Care must be taken though to not completely block the device if it's traffic patterns are normally irregular. Image for example a security camera that only starts streaming on request. The system as currently implemented would gather that this was a low-traffic device, and the sudden increase in traffic from starting a stream would likely be flagged as bad. If the QOS algorithm were to sanction it too hard it would render the stream, and thus the device itself completely useless. Honouring both these requirements, both to block bad data that is actually bad, and not stopping actual data mislabelled as bad would be difficult to do properly.

However a regular stream would likely be less affected than a DOS attack for multiple reasons. First of a DOS attacks value is directly proportional to it's bandwidth. Limit bandwidth by 10% and the DOS attack is limited by 10%, of course the receiving end have a pain-point at which it simply wont be able to handle more traffic so as long as the attack stays above that threshold it

can be said to be 100% effective. But the available targets are diminished proportionally with the available attack strength. Normal traffic like a stream on the other hand is often adaptive to it's allotted bandwidth so given less bandwidth the stream would still work, albeit the quality might not be as good. This holds true for most good traffic. However dialling down the quality of normal data is a very severe penalty for a system like this and should be avoided if at all possible.

6.4 The field of DDOS protection

As mentioned in the introduction there are many ways DDOS as a problem is being treated. Most of these however are server side and aim more at being able to handle the amount of traffic instead of avoiding it. This solution tries to limit the usability of DDOS as an attack in itself, and not only for those who are able to pay for and set up server-side DDOS protection. It doesn't rely on, or interfere with, any of the existing solutions for DDOS prevention and can therefore be seen as a compliment to those.

Another benefit of this system over other traffic learning devices like the DoJo by Bullguard[12] is that this system could be implemented into existing router firmware. Preventing DDOS attacks is not something a small group of people can do, it requires the majority of devices to be secured. By installing systems like the one described in this thesis, either by free will or by legislature, on most consumer routers could severely help in deteriorating the usability of DDOS attacks.

7

Conclusion

The system displayed in this thesis offers a novel approach to DDOS mitigation which not only benefits those that use it, but everyone affected by DDOS. During the design, implementation, and testing many interesting problems have surfaced and been discussed throughout the thesis. The implementation offered along with this thesis offers a base upon which a functional system can be built to provide the benefits of the design. Defeating the problem that is DDOS is something which should be handled not simply by treating the symptom on the server side, but something that should be cured by better securing our devices. While IOT devices lack behind systems like the one described here would be able to offer a first level defence against the threat.

Further research into better low-resource systems for learning network traffic would be able to be added as drop-in replacements for the current learning algorithms to make the implementation more viable. But a certain baseline and proof-of-concept has been provided in this thesis. Not inconceivably it would also be possible to choose completely different metrics, or simply add more metrics to the system to attain better results.

The low-memory consumption per device and likewise low amount of operations to perform per packet makes this system quite readily ported to hardware with strict limitations on both. It is also completely void of floating point calculations, except from timekeeping in the graphing utility. This is something which would heavily penalise devices without hardware floating point operations. These measures are important as reusing the same hardware as used today

would help keep prices low for such products and allow a higher adoption rate of these systems.

Bibliography

[1] Internet Live Stats. Number of internet users (2016).

[2] Pew Research Center. Majority of internet users in most countries are daily users.

[3] Cisco Visual Networking Index. The zettabyte era: Trends and analysis.

[4] Dyn EVP Scott Hilton. Dyn analysis summary of friday october 21 attack.

[5] Network Working Group P. Ferguson. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing.

[6] Jakub Czyz, Michael Kallitsis, Manaf Gharaibeh, Christos Papadopoulos, Michael Bailey, and Manish Karir. Taming the 800 pound gorilla: The rise and decline of ntp ddos attacks. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, pages 435–448, New York, NY, USA, 2014. ACM.

[7] Christian Rossow Thorsten Holz. Horst Gortz Institute for IT-Security Ruhr-University Bochum Germany Marc Kuhrer, Thomas Hupperich. Hell of a handshake: Abusing tcp for reflective amplification ddos attacks.

[8] Senior Reporter SC Magazine Bradley Barth. Zscaler traffic analysis finds iot devices misbehaving.

[9] Alan Saied, Richard E. Overill, and Tomasz Radzik. *Artificial Neural Networks in the Detection of Known and Unknown DDoS Attacks: Proof-of-Concept*, pages 309–320. Springer International Publishing, Cham, 2014.

[10] Cloudflare. Ddos protection.

[11] Imperva Incapsula. Ddos mitigation.

[12] BullGuard. Dojo by bullguard.

[13] Cristina Iordache and Ping Tak Peter Tang. An overview of floating-point support and math library on the intel "xscale" architecture. In *Proceedings of the 16th IEEE Symposium on Computer Arithmetic (ARITH-16'03)*, ARITH '03, pages 122–, Washington, DC, USA, 2003. IEEE Computer Society.