University of Amsterdam

System and Network Engineering

Research Project

Controlled DDoS Security Testing

Authors:
Mike Berkelaar
Azad Kamali Roosta

July 11, 2014

Abstract

This report describes the concept of performing controlled (distributed) denial of service (DDoS) attacks for auditing purposes. With DDoS mitigation strategies and mechanisms likely not being present in a test-setup the only option may be to perform these attacks on a live production system. However, this introduces the danger of causing severe damage in the form of (partial) unavailability. Therefor we propose a method of doing a controlled attack where the effects are closely monitored to prevent this damage from occurring.

We researched mechanisms to monitor general forms of attacks from the application- and network-level attack classes. For application-level attacks we propose a prediction algorithm that prevents the attack from growing too big based on the slope of the response times being monitored over time. For network-level attacks we propose a high frequency monitoring method that approximates the global packet-loss rate. These mechanisms have a proof-of-concept implementation in Python for testing purposes.

Contents

1 Introduction

A (Distributed) Denial of Service ((D)DoS) attack is mostly done by consuming a target's resources in a way that it is no longer able to provide its services as intended or in the desired way. These resources can be real hardware resources, like available computing cycles, or logical resources like the amount of simultaneous open connections a server can handle. Other type of denial of service attacks would disrupt the service by either blocking the communication or altering the physical state in a negative way.

During this project we will be focusing on the resource demanding attacks, commonly found in distributed denial of service attacks. With this method an attacker has to gather enough resources to outweigh the target or misuse a vulnerability that will exhaust the resources in a more efficient way.

In this project, we will be establishing a framework, along with a proof of concept, which allows different systems to be tested in various simulated DDoS scenarios. The simulation should be powerful enough to show the effect of the attack as well as the systems reaction to it, but at the same time should be managed in a way that does not cause an actual DoS as it may be used on live environments. The results of these tests could be beneficial to audit the DoS resiliency of an environment and show potential bottlenecks.

The resource demanding attacks also have the characteristic of being hard to distinguish from normal traffic. From the prospective of intermediate devices, they are not that different after all. With that being said, the concept of a simulation for such attacks would be quite close to load testing and stress testing. To be more precise we can call it a combination of these two tests happening at the same time on a live system with a vital sign monitoring system which acts like a safety valve, preventing the DDoS attack from actually doing damage.

1.1 Research question

Simulations are a good way of testing things in practice. If everything is working perfectly on paper, it does not necessarily mean that they will actually work as intended. Evaluating the readiness for attacks on the infrastructure is a critical part of any IT security audit. In case of DDoS security testing we may want to evaluate the performance and effectiveness of DoS security measures in place.

Penetration- or stress testing is the main way of doing such audits. While checklists and process reviews may give one an indication of how secure an infrastructure is, these tests can be seen as a method which will reveal if all efforts enable the infrastructure to work as intended. Most of tests can be done without a high risk of causing damage, hence they can even be performed over a live service. A Denial of Service test however is an exception to this as the goal of a DoS attack is to prevent the target from providing its service.

The research question central during this project is:
- *How can various DoS attacks be simulated in a controlled way?*

Essential to the research question are a number of subquestions:
- *Which DoS attacks can be simulated in a potentially controlled way?*
- *Which parameters should be used in order to have a controlled attack?*
- *Which metrics should be monitored to measure the effects of a DoS?*

1.2 Related work

The replication of denial of service attacks for testing and auditing purposes is an active field of research.

The concept of load testing a service has been researched by S. Abu-Nimeh et al. With a number of selected tools and metrics an application can be tested to show its bottlenecks. It is suggested that with these results the corresponding configurations can be optimized in order to be prepared for high demanding scenarios like a denial of service attack or flash-crowd. [2]

S. Bhatia proposes a framework to realistically simulate DDoS attacks in a distribute fashion and gather performance metrics from this action. A custom traffic generator, *Botloader*, was created to perform different attack scenarios with clustered capabilities. [1]

M. Yan et al have researched the load testing of webservices on the application level. Here it is shown that various benchmarks from a single source produce different behavior from the same load coming from a distributed *TaaS* (Testing as a service) solution. 'WS-TaaS is the proposed solution for leveraging cloud infrastructures to create a load test that is more comparable to real usage.[3]

2 Background

There are numerous denial of service attack methods in existence that are maliciously performed to degrade the performance or availability of a service. During this research we distinguish attacks that target the available bandwidth, or network-level attacks, and application-level attacks that target the over usage of expensive functions in these services.

Network-level attacks over consume the available bandwidth between the source and destination by flooding it with unwanted traffic. The effects of this flood could saturate the links between legitimate sources and the DDoS target anywhere along the path, resulting in an interrupted availability.

Application-level attacks misuse expensive functions or characteristics of a service to cause resources to be rapidly consumed on the DDoS target. This could be caused by out of ordinary usage of for instance a search function, which is likely to depend on a lot of backend queries that could be rather complex and resource intensive. Other sorts

of application-level attacks could target certain weaknesses in protocols, like keeping connections open by performing artificially slow requests and responses.

Despite the complexity or volume of some DDoS attacks there are mitigation methods available like filtering of traffic, blacklisting of malicious sources or even just optimising configurations of the service. However, testing the effectiveness of these mitigation methods is hard without the risk of causing damage by uncontrollable traffic generators that can simulate a DDoS attack.

3 Approach

Our focus during this research is exploring possible methods of performing controlled attack simulations based on the analysis of remote observables. One of the key dilemmas in being able to control the attack is to determine when an attack is successful in the sense that a malicious attacker would reach its goal of disrupting a service. Defining the point that a service is disrupted or damaged may seem straightforward, but the tolerated level of availability may differ hugely between the type of services, protocols used, their usage and application.

In any audit process a clear goal has to be set which allows the tester to eventually determine if a target has passed the test successfully or not. Usually this goal is expressed in a threshold which the target is supposed to handle. As an example case, consider a scenario in which a password needs to be secure from brute-force attacks. With a defined threshold of six months we would state that the scenario would fail if it would take less time to crack, and pass in case it was more.

In a DDoS audit these thresholds may be harder to define. With most DDoS attack classes the target is the availability of a service, which can often only be measured as either available or not. Depending on the scenario it may be the case that the measurements show a disrupted service by means of extra delay or even lower actual availability with a number of requests simply not being processed. These situations have totally different impacts on cases like online banking or name resolving, where the latter is far less sensitive to incidental disruptions.

In order to come up with a comprehensive way of performing controlled attack simulations it is important to find methods to observe a DDoS attack which can be applied on both ends of this spectrum. The concepts researched during this project will be implemented in a proof-of-concept implementation, discussed further in this document.

4 Basic concepts of a controlled DDoS

In order to analyze and test for DDoS resiliency a possibility would be to simulate an actual attack, but without crossing the point where damage would occur. We set out to find a way of performing such attacks where the effects are, to some degree, controllable and therefor less risky to perform.
During this research we will consider a DDoS target as a single entity, exposing services over a common protocol like HTTP.

In order to come up with a method to perform such a controllable DDoS attack the following sub-questions were considered:

- What are the direct effects and observables of a DDoS attack?
- How can DDoS attacks be reliably controlled based on these observables?

4.1 Attack observables

With both network- and application-level DDoS attacks it is required to find observables that are a direct effect of the attack. If these observables are in any way consistent or predictable then they might be useful to provide feedback to the attack simulation.

Network-level DDoS attacks are likely to show signs of packet-loss along the network path once the volume of the attack congests the links to the target. W. Su et al illustrate how this behaviour can be monitored and utilized to trace attacking sources during a DDoS attack [5]. With the links congested somewhere along the path there is a direct relation between the capacity of the link and the volume of all normal and attack traffic combined, resulting in a decreased probability of every packet actually reaching its destination.

Application-level attacks may result in local computing resources to be exhausted, leading to unexpected behavior that could be very specific to the targeted service. However, we believe that a number of generalized scenarios can be expected. With less available resources an increasing latency on responses is to be expected. In some situations it may be the case that the service has to selectively process requests and drop the remainder, as the outstanding queue would grow infinitely.

4.1.1 Expected behaviour

As mentioned earlier, the idea is to try to attack the target while keeping an eye on its operational status and vital signs. These could be any of the resources being monitored, based on the attack type. We expected that as the attacking rate would increase in small steps, so would the vital signs like the response time of the target. This way, we could set a point at which the system's operation is not acceptable by the end user anymore, a defined threshold, yet has not reached the point of absolute unavailability. Figure 1

shows a representation of how these vital signs would be monitored during the attack simulation, grouped as 'feedback'.

Figure 1: Expected feedback

However, the expected increasing values of the monitored resources, like the response time of the service, showed to be everything but smooth and gradual. The reason for this is likely to be different for each resource type.

For instance, if we consider the ICMP round trip time (RTT) or HTTP response time being the feedback variable and the network bandwidth to be the targeted resource then we expect the lowest link capacity to run out at some point in time. When this is the case the packets used to measure the RTT will start to get a lower probability of actually reaching the target. Another observation may be that a certain level of queueing happens at this point anywhere along the path to the target, introducing the slightest of extra delay, although this is likely to be unnoticeable as the extra delay may only be a fraction of the total RTT. With low monitoring frequencies the observable RTT will therefor show sudden signs of packet-loss, as illustrated in figure 2.

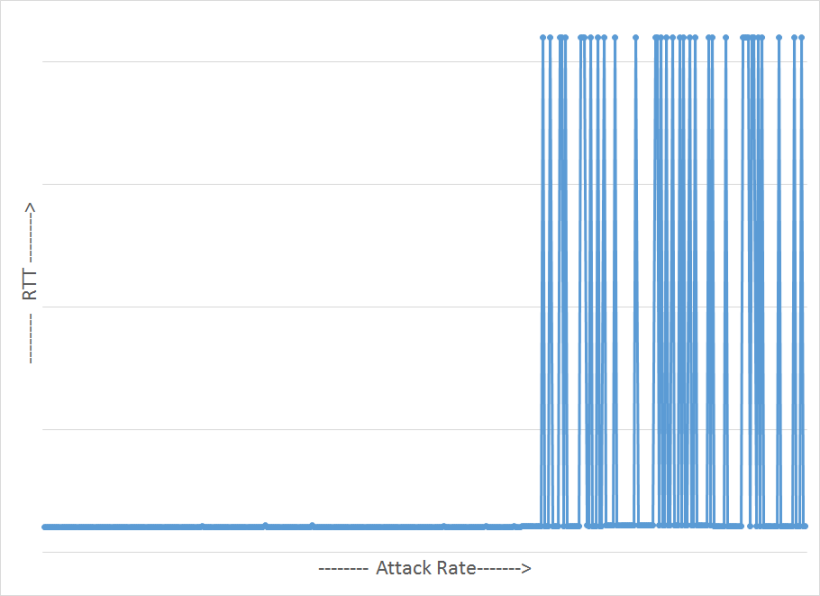

Figure 2: ICMP monitoring RTT/Time-outs

An interesting phenomenon found is that although the RTT will not increment according to the growth in attack rate, it will show a very distorted development to the point of essential unavailability, whilst it is very stable till the point of convergence. The concept of keeping track of the variance in RTTs is something that may show potential, although it is still hard to approximate the point of inoperability or the defined threshold as illustrated in figure 3.

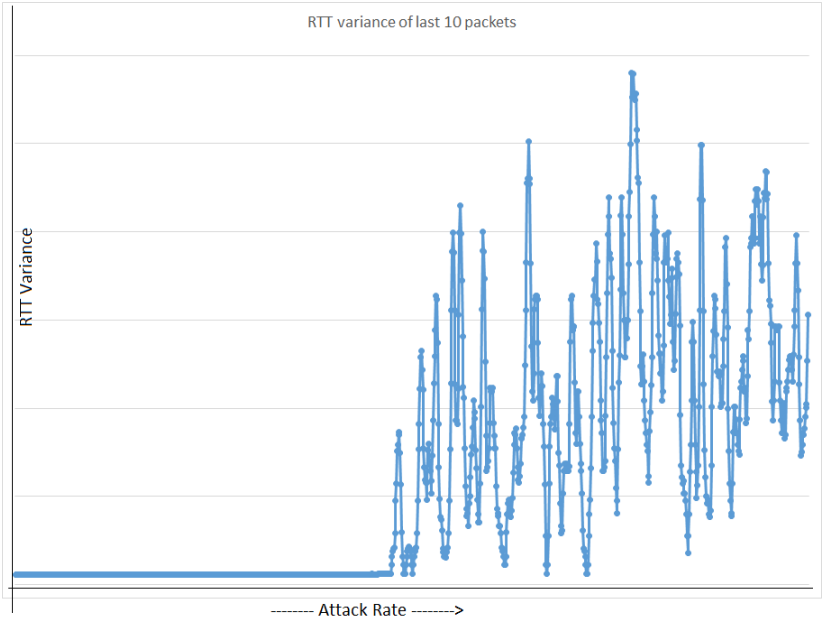

Figure 3: RTT variance

The other parameter candidate as the feedback variable is the frequency of timeouts. As stated earlier, RTT times tend to be either normal or timed-out (lost). With relative high frequency monitoring the number of time-outs are related to the probability of a single packet being forwarded when the capacity of a link is reached. Based on this type of feedback it may be possible to approximate the total packet-loss during a simulation. As some degree of packet-loss is acceptable even in normal operation we will have to further look at what thresholds are suitable for this. Figure 4 shows the result of high frequency monitoring at the moment of an attack simulation causing packet-loss.

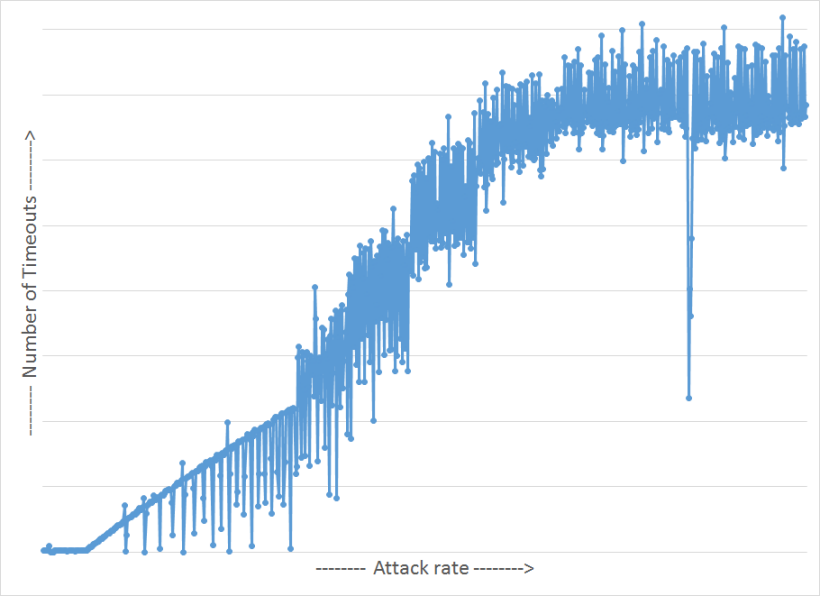

Figure 4: Timeout Frequency

In case of an application level attack the network bandwidth resource is often totally irrelevant compared to the local resources. We found that monitoring the local resources with daemons like SNMP is not accurate enough. We also fear that the performance and granularity of these daemons would decrease with an ongoing attack simulation as resources run out. Rather than finding different ways we revisit the previously discussed methods on the response time and number of timed out requests. We observe that the response time of an HTTP service increases more gradual to the point where a threshold could be placed, as resources run out less abrupt. An illustration for the behaviour of the application level response time is included in figure 5.

As can be observed, when the server is working under normal workload, most of the requests are being answered within a relative amount of time. But as the load increases, some of the requests, although not all, are being delayed. The growth in responses does not correspond to the increment of requests one by one. Instead, at a certain point we will experience a jump in response times.

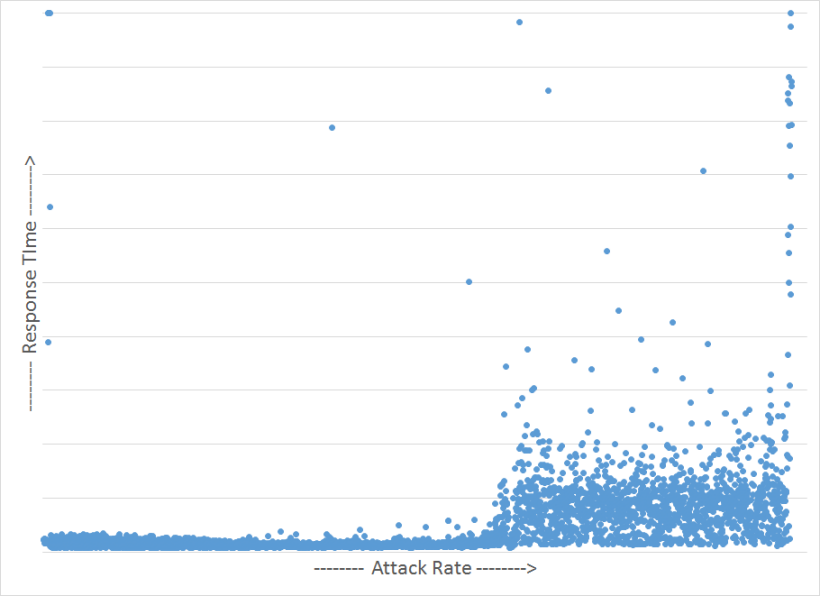

Figure 5: Distribution of HTTP Response Times over time with increasing attack rate.

4.2 Monitoring aspects

The concept of using a monitoring system to provide feedback may seem straightforward at first. However, as we saw in the previous sections there are a number of concerns to address:

- Choosing the right sensors
  Different attacks target different resources of the victim. Hence a victim's vital signs are interpreted differently based on attack type. Although keeping an eye on all available resources could have advantages, it may be the case that only a limited amount of them can provide the proper feedback required for a controlled attack simulation.

- Real-time monitoring
  The whole idea of having a feedback loop is so that the attacker can regulate its attack traffic according to the feedback gathered from the target machines. It is quite obvious that outdated monitoring values would result in false and unreliable feedback, defeating the whole purpose.

- Out of band feedback signaling
  One important point with regard to monitoring traffic is that it is best if the attack traffic could in no way interfere with it, or the monitoring data could end up skewed or delayed. In some cases we may actually want the attack to influence the monitoring data in order to draw conclusions based on for instance monitoring availability.

- Monitoring watchtower location
  When trying to monitor the vital signs another debate is where the actual monitoring should happen. Some values, like the local resource utilization, can only be obtained from the local system while observable values like the response time may better be distributed to multiple monitoring agents for improved accuracy. Also, monitoring resources from the local system itself may show troublesome when monitoring data can no longer be monitored due to the unavailability of spare resources during an attack.

4.2.1 Thresholds

In some cases simple thresholds can be set to define the boundaries that the vital signs of a service can be measured at during an attack simulation. In cases like the response time these are likely to be different between services. Live environments of distinct applications will have different allowances on the speed and responsiveness, like comparing a webmail environment to a stock trading application. Therefor we believe that these thresholds should be taken as user-input for every test performed. However, we can assume this threshold to be measured in a number of seconds.

In other cases the response time is not a very good monitoring value to control an attack simulation, as discussed in section 4.1. For instance, with a network-level attack we may only observe the frequency of packets being lost. A number of packets being lost does not directly translate to a certain level of (un)availability of a service as used protocols may have different ways of handling recovery mechanisms, like retransmissions. The effects of packet-loss on TCP communication is an extensive field of research. T.V. Lakshman et al analysed the effects of random loss in communication continuity [4]. We performed a test to analyse the effects of packet-loss on HTTP communication in our proof-of-concept, described in section 6. In this test we monitored the success rate of HTTP requests with packet-loss rates of 0.1, 1 and 2 percent of the total traffic. Figure 6 shows that the higher packet-loss rates result in much more HTTP requests having a higher response time or even timing out due to retransmissions, indicated by the upper boundary of the line. These values require more research to be used as thresholds to measure the general availability of a service under a DDoS attack. We believe that different values may be required for different protocols and link capacities, although they can be used to test the concept of controlling an attack simulation with this method in our proof-of-concept setup.

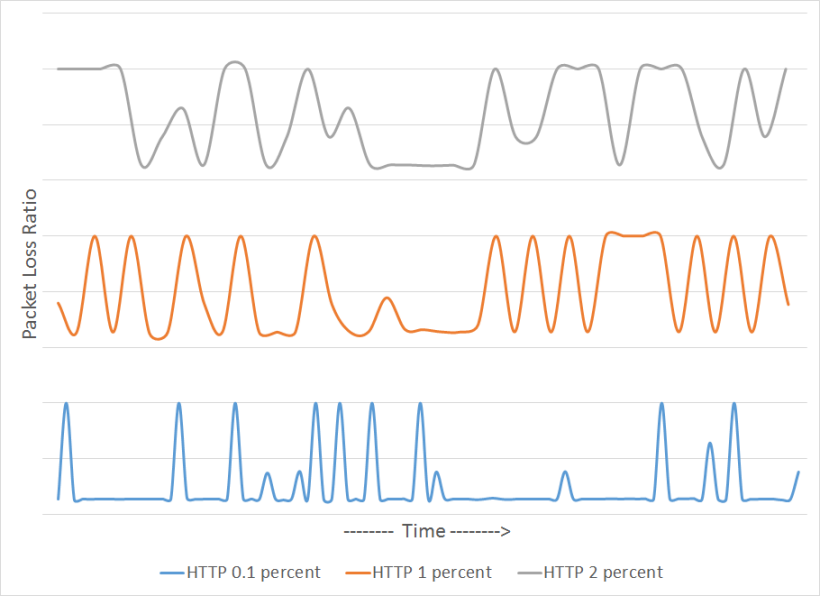

Figure 6: HTTP response time and loss

4.3 Controlled attack

In order to simulate attacks a traffic generator is required that can generate packets equal to a malicious DDoS attack of its class. An important addition however is the possibility to have precise control over the effect of the simulation.

Both network- and application-level attacks can be measured in a number of units per second. In case of a network-level attack the number of packets per second (PPS) and their size define the volume of the attack. An application-level attack usually consists of a number of transactions per unit of time, making up requests in some form or way at a packet rate not necessarily equal to the request rate.

Both attack classes can be controlled at the most basic level by adjusting the volume of the attack based on these parameters. A method of doing a controlled attack simulation could be to increment the volume based on monitoring the observables as discussed in section 4.1. If the monitoring data shows no sign of the attack hitting a threshold or causing damage then the attack rate can be increased. If the last or next increment however is predicted to do so then the attack should be halted.

The increments should be interleaved with periods of a constant attack volume in order to get reliable monitoring feedback with as little variation due to the simulation itself. As the monitoring feedback is expected to show only limited change till the DDoS is about to cause damage, the increment rate is considered to be a constant for the duration of the simulation. A variable increment rate would likely make options like trend monitoring or predictions a lot more complex, due to this behaviour. The increment should be based on the approximate capacity and the permitted aggressiveness of the

simulation. While lower increments would result in a higher granularity, it may actually result in the simulation being skewed for being too slow, as variations in monitoring feedback is to be expected on a live system over longer periods of time.

5 Proposed solution

In order to do a controlled DDoS simulations we propose a framework that is extendable with various DDoS attacks. This framework is separated in a number of components regarding the traffic-generation, monitoring and orchestration of the DDoS attack. This chapter describes the architecture and design of the methods of section 4, while a proof of concept implementation is discussed in section 6.

5.1 Architecture

The traffic generation of the attack is a resource intensive task and may require the combined strength of multiple nodes in order to test more powerful services. It may also be of added value to gather monitoring data from multiple nodes in different networks, resulting in different network paths to the targeted service being probed. These distributed components have to be orchestrated by a central entity we refer to as the Master. While the agents only follow instructions and pass on the gathered data, the Master actually performs the logic to control the attack simulation in a controlled way. A visualisation of the architecture is added in figure 7.

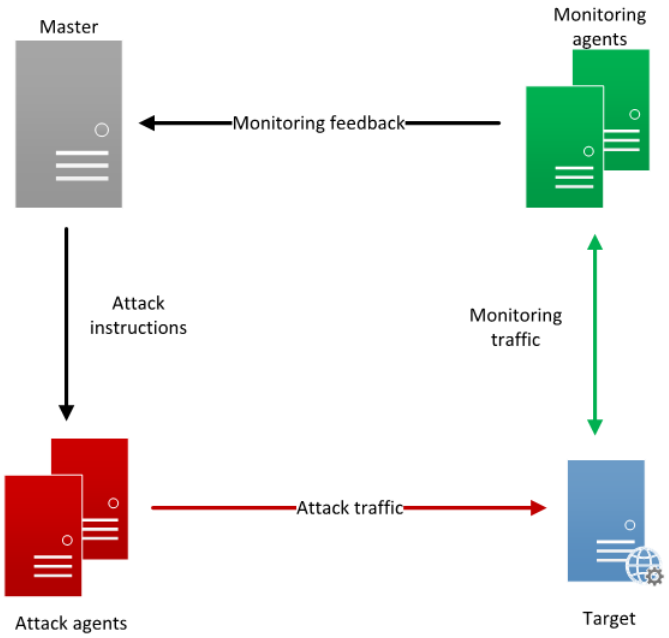

Figure 7: Architecture

An essential part of this architecture is the communication between the agents and the master. Monitoring data has to propagate to the master as quick as possible in order to determine if the attack simulation reached, or is about to reach, a threshold. If the

attack simulation has to adjust certain characteristics of the attack, or abort it entirely, then these instructions have to be spread to all attacking agents as soon as possible in order to avoid the damage discussed in section 4.1. Apart from that it is even more important to have these delivered at all and without corruption. Therefor, all agents set up TCP sessions to the master, ensuring reliable delivery and a certain degree of data integrity.

5.1.1 Messages

A simple message format was defined for all agents to communicate with the master. We suspect that extending this framework with different attack classes will also require different parameters and monitoring values over time. Therefor, an easily extendable message format is used based on key-value pairs.
Another aspect that was handled in the message format is the issue of an attack agent becoming unresponsive. It may be possible that during a simulation the agent consumes all of its own capacity and is therefor no longer able to communicate with the master, resulting in the attack proceeding without the master still being in control. To handle this we propose a 'ticket' mechanism where the attack agent is instructed to perform the attack during a certain time slot and stop if no further instructions are received, preventing total loss of control.

The formats used for the monitoring- and attacking agents are shown in listings 1 and 2.

Listing 1: Attack format

```
id: {
    Hostname: Target_Hostname
    Port    : Target_Port
    Attack  : {
              Type    : Attack_Type
              Rate    : Attack_Rate
              Ticket_start : Start_time
              Ticket_end   : End_time
              Param_A : A
              }
    }
```

Listing 2: Monitoring format

```
id: {
    Hostname   : Target_Hostname
    Port       : Target_Port
    Monitoring:   {
                  Type      : Mon_type
                  Time      : Timestamp
                  Mon_data  : {Data}
                  }
    }
```

5.2 State-machine

The attack simulation can have a number of states that are defined in the state diagram of figure 8. We envision the monitoring- and attack agents registering at the master and listening for instructions. The master initialises the agents by sending the parameters of the attack and monitoring. Based on the processed monitoring data, or feedback, the attack will be regulated to either stop, increase, decrease or keep the rate constant.

As explained in section 5.1.1, we limit the validity of these instructions. In case an agent becomes unresponsive it will only perform an attack for the remainder of the instruction's validity.

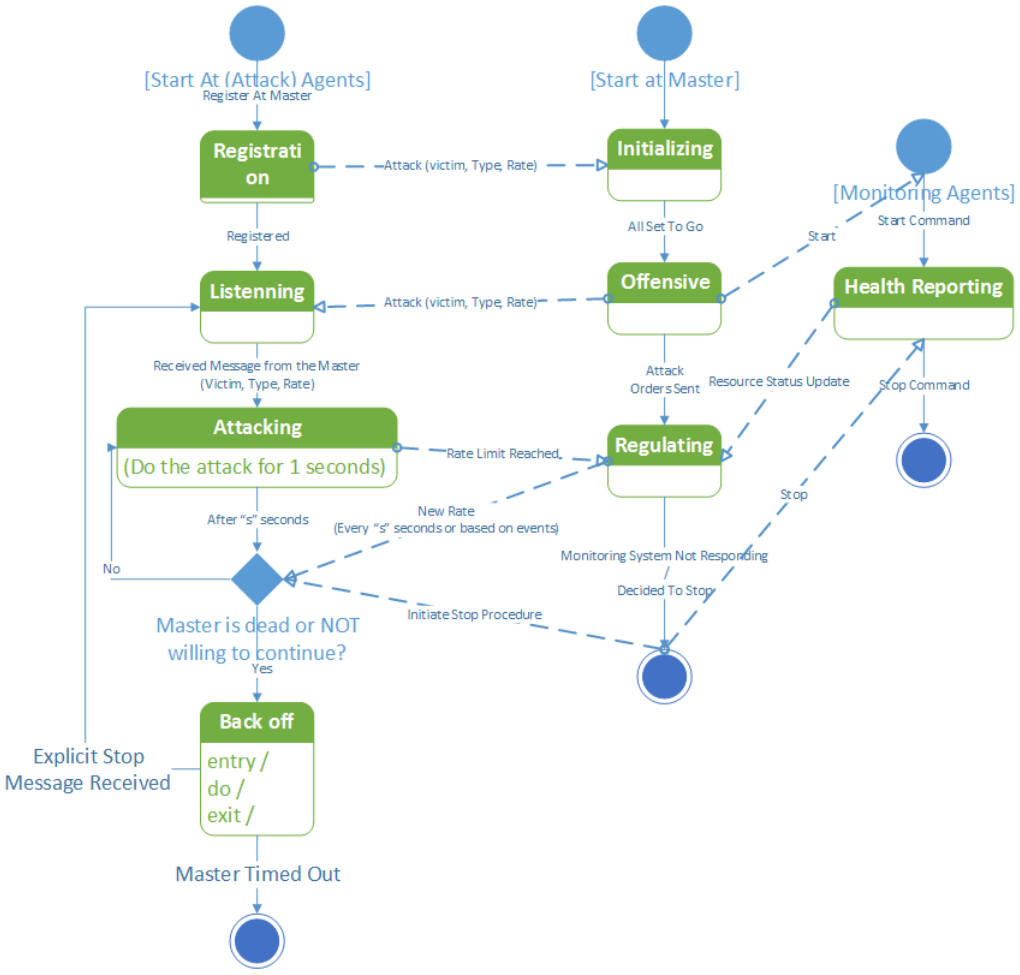

Figure 8: State machine interaction

5.3 Monitoring methods

The observables during an attack simulation were analyzed in section 4. The monitoring data that we will be focusing on are the remote observables of a target with ICMP and HTTP packets. The probing for this data is handled in the monitoring agents. The interpretation of this data is done by the master, which translates this into new instructions for the attack agents.

5.3.1 Probing

A probing method that was considered is the use of local monitoring daemons that could precisely report on the resource utilization. Although this method could have favorable characteristics for application-level attacks it was decided to not further research this option as we think that the performance and granularity of these daemons would de-

crease with an ongoing attack simulation as resources run out. Preliminary testing also showed that some of these daemons, like *SNMPd*, do not show real-time monitoring data but rather a sliding average of roughly the last minute.

The observables discussed in section 4.1 have in common that they can be gathered from remote and are directly influenced by the availability of the target. The metrics that we will be focusing on are the ICMP Echo round trip time (RTT), ICMP Echo packet-loss count, HTTP response time and HTTP packet-loss count.

These metrics can be measured by sending ICMP Echo- and HTTP-requests, and keeping track of the replies. In order to reliably interpret the results of these measurements we need to make sure that the requests adhere to some expectations. For instance, the intervals between these requests should be evenly spaced to prevent the probing from showing bursting behavior or creating large unmonitored gaps because of unexpired time-out values, as this would likely skew the response times and loss count. These concerns become even more important when the probing rate should be increased for a higher granularity. An asynchronous probing solution would allow evenly spaced requests to be sent at a constant rate while incoming responses are then handled separately.

A network-level DDoS attack will cause congested links to the monitored target. As a result the probing requests have a rather large probability of being lost, suggesting packet-loss is occurring but not at what rate. In order to approximate the packet-loss rate one solution would be to increase the probing rate to a substantial portion of the total traffic. With a known ratio of the probing rate opposed to the attack simulation volume we can approximate the global packet-loss rate and compare this to the thresholds defined. A requirement for this however is that the probe packets should be equal in size to the packets of the attack simulation, or else no comparison could be made on solely the PPS. In the proof-of-concept implementation a ratio of $95 : 5$ was utilized, resulting in the monitoring actually making up roughly five percent of the total attack [6].

5.3.2 Monitoring logic

The probing data resulting from the methods discussed in section 5.3.1 will require different ways of interpretation. Despite the different approaches they all have in common that the processing should happen in near-realtime in order to provide rapid feedback to the attack simulation.

By watching the slope of increasing response times or occurrences of packet-loss over time, certain prediction could be made. The approximation of the total packet-loss count and monitoring for hard thresholds are, on the other hand, more practical approaches to check if an attack simulation is considered to be at its limit or if there is still a margin to continue.

Slope watching

In order to proactively monitor the feedback received from the target, one needs to estimate when the threshold would be reached. One way of doing so is to look at the

very last sample and trigger an alert as soon as it gets close to an specified boundary or threshold, or as soon as it finds an extra ordinary gap with the previous sample. However, relying on a single sample is likely to show a lot of false positives. A way of monitoring the trend of samples is needed instead of relying only on the last value. By investigating different behaviors we found that in some cases, like HTTP response times, such a trend can be mapped to the slope by which the last group of samples are either de- or increasing. For this purpose we measure the slope of a logical line crossing the current sample and the first sample after latest change in direction of samples. So as soon as the samples start to ascend/descend, we mark our first sample and when the direction (ascending/descendin) changes again, we mark our last sample and calculate the slope of line crossing these two. The resulting slope value can then be compared to a defined threshold. However, as the samples do not necessarily steadily de- or increase we need a way of finding high deviations in the monitoring data. The First_Sample would always hold the value of the first sample after a sudden change in the slope. The Current_Sample is the current value that the master received to process, while the Last_Sample is the value of the previous monitoring value. The following pseudo code describes the algorithm that would be called for every received monitoring value by the master:

Algorithm 1 Slope watching

```
if ((Last_Sample ≥ First_Sample) AND (Current_Sample ≤ Last_Sample)) OR
((Last_Sample <First_sample) AND (Current_sample >Last_Sample)) then

    # Swap Direction
    First_sample = Last_Sample
    Count = 1
else
    Count = Count + 1
    if ((Current_Sample - First_Sample) / Count) >Threshold then
        Trigger_Alarm()
    end if
end if
```

Packet-loss approximation

With the attack simulation likely making up the better portion of the total traffic it is possible to make relatively accurate approximations of the total packet loss rate. A possibility of doing this is by calculating the ratio of lost ICMP probing requests or responses compared to the total traffic sent to the target over a period of time. As the monitoring probing rate has a sliding value fixed to the total attack simulation rate this value always scales to the defined treshold. The following formula calculates the total packet-loss ratio that can be compared with the threshold set for the simulation, with the ICMP probing rate (P), received ICMP responses (R) and attack simulation rate

(A):

$$Loss = (P - R)/(A + R)$$

Hard thresholds

Another practical method would be to regularly check if certain thresholds, like the ICMP RTT or HTTP response time, are reached yet. Although this is relatively safe way of monitoring an attack simulation, it is also likely to trigger too fast due to incidental noise or peak values in the probing data. We expect the probing values to show a distorted incline before being passed through a smoothing function. In order to filter out these outliers we define a function that won't trigger before at least half of the last (X) values equal the threshold, where X should be a reasonable number based on the monitoring frequency.

Algorithm 2 Hard threshold - Mean

```
X = 20
while ProbingData do
    if Value > Threshold then
        Y = Y + 1
    end if
    if Value < Threshold then
        Y = Y − 1
    end if
    if Y >= X then
        Exit
    end if
end while
```

5.4 Attack simulations

For this research project we made the distinction between application- and network-level DDoS attacks. We found that these attack classes in general require different ways of monitoring in order to prevent damage.

Network-level

With a network-level attack we target the available bandwidth to a service by flooding it with traffic of any type. As discussed in section 4.1, a good observable for this attack class is the number of packets lost during the attack. To monitor the attack simulation we previously defined the packet-loss approximation method in section 5.3.2. With this method the monitoring frequency is a sliding percentage of the total attack volume, really being a part of the attack in itself. One can then detect the occurrence of packet-loss based on the number of returned or acknowledged monitoring packets and compare this to the defined threshold, given that the monitoring packets are equal to the attack packets.

Figure 9 shows the attack simulation consisting of the attack-rate and monitoring combined. The actual echoed or acknowledged monitoring packets may be lower than the monitoring packets sent when packet-loss occurs.

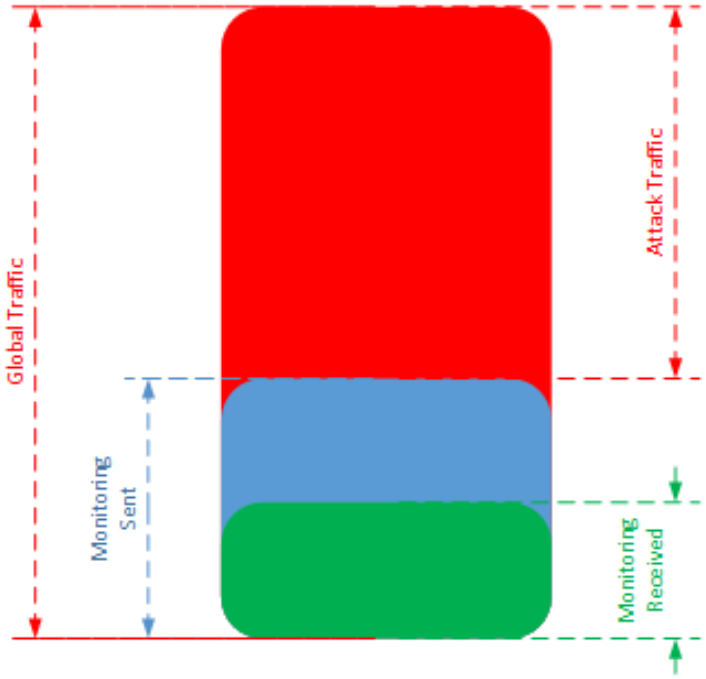

Figure 9: Traffic flood components

Application-level

An application-level attack targets the local resources of a service. By mis- or overusing certain characteristics of a service one exhausts the resources, resulting in an increasing response time as the processing of incoming requests is being delayed. The main observable of this event happening is the response time and variance between successive requests increasing. We believe that the slope watching method from section 5.3.2 can predict the point of reaching a threshold in advance by analysing the increments of the response time over a longer period of time. This could be an advantage as the threshold that is considered as damaging can be avoided altogether.

6 Proof-of-concept

In order to analyze the methods of controlling DDoS attacks we created a proof-of-concept implementation of the architecture discussed in section 5. This implementation is coded in Python and consists of the distributed architecture of attack- and monitoring agents and the master. The source code to this project is published on GitHub. [6]

We tested the scenarios of both a network- and application-level attack with generalized methods. Although the implemented attacks are in no way advanced or performant, they do enable us to test the effectiveness of the monitoring methods used to control the attack simulation.

6.1 Experimentation setup

The proof-of-concept was tested in a shared environment which required some adjustments to the maximum attack volume during our tests. We created a test setup consisting of a Linux server based on Ubuntu to facilitate an HTTP service. We used different Linux machines to perform the tasks of attack/monitoring agents and the master. Although these machines are all interconnected at speeds of at least 1Gbps, we configured the server's interfaces to 100Mbps in order to prevent a denial of service from growing beyond just the targeted server during these tests.

An extension to the proposed architecture for these tests is a control panel that directly hooks into the monitoring data that is gathered on the master. It allows manual intervention of the attack simulation while underway to also test beyond the threshold values defined at the start of the test. The code to this is also in the code repository. A screen capture of this tool is shown in figure 15 in appendix.

Using this panel, the operator can see the overal statistics about monitoring system, feedback and also attack parameters like current attack rate. The operator also can intervene at any time and change the rate of the attack using the slider.

6.2 Network-level: Traffic flood

The traffic flood attack performed sends fixed size packets of 500 bytes to the victim machine at a rate controlled by the master. The monitoring of this attack is based on the packet-loss approximation of section 5.3.2, making up roughly five percent of the total attack-rate with equal 500 byte ICMP packets.

Instead of increasing the attack rate divided over all the distributed attack agents we increase the monitoring rate above five percent of the threshold to get direct feedback on whether this would result in packet-loss. If an increment of the monitoring rate is not triggering any of the defined thresholds the monitoring rate is reset to five percent of the attack rate, while the attack rate is increased with the last increment. We refer to this process as the 'hand-off' procedure. Figure 10 shows the monitoring rate (green) and acknowledged monitoring rate (black) increasing equally, suggesting no packets are

being lost. The attack rate (red) is increased every hand-off sequence.

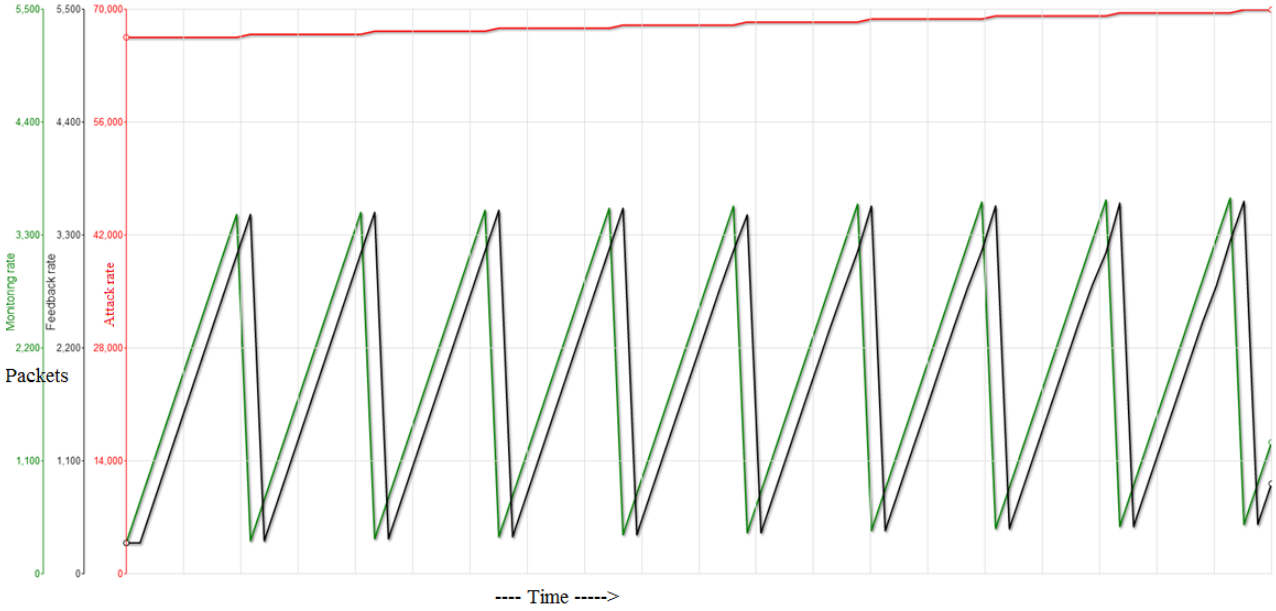

Figure 10: Traffic flood process

If we look at the results from a full traffic flood simulation, illustrated in figure 11, we can see that at some point packet-loss starts to occur, visualised by the difference between the green (monitoring rate) and black (acknowledged monitoring rate) lines. This simulation was configured to stop when the monitoring approximates that 1 percent packet-loss is occurring for the global traffic to this target [4.2.1]. This threshold is met at the end of the graph, after which the simulation is sustaining the attack rate in a stable state.

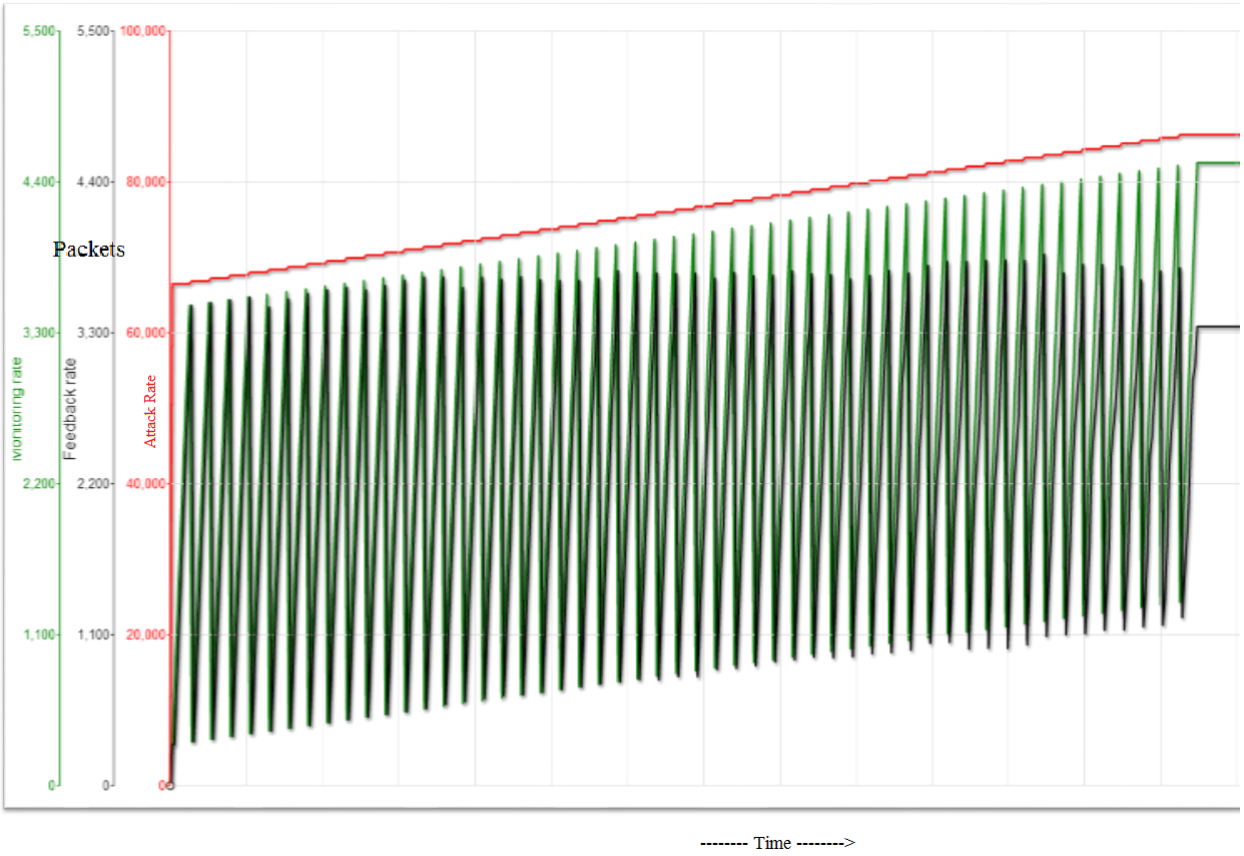

Figure 11: Traffic flood (full)

6.3 Application-level: HTTP flood

The application level attack is similar to the traffic flood in concept.
The attack was ever increasing with 10 requests every 5 seconds. As it can be observed in Fig. 12, the response time (red) started to increase and became variant when roughly 500 HTTP requests per second are being made (green). The simulation continues until the slope (blue) threshold is reached at 100. At this point, the Master predicts that an outage is imminent if the attack simulation is continued and orders the attack agents to halt. The attacked service is able to quickly recover from the attack once it is stopped.

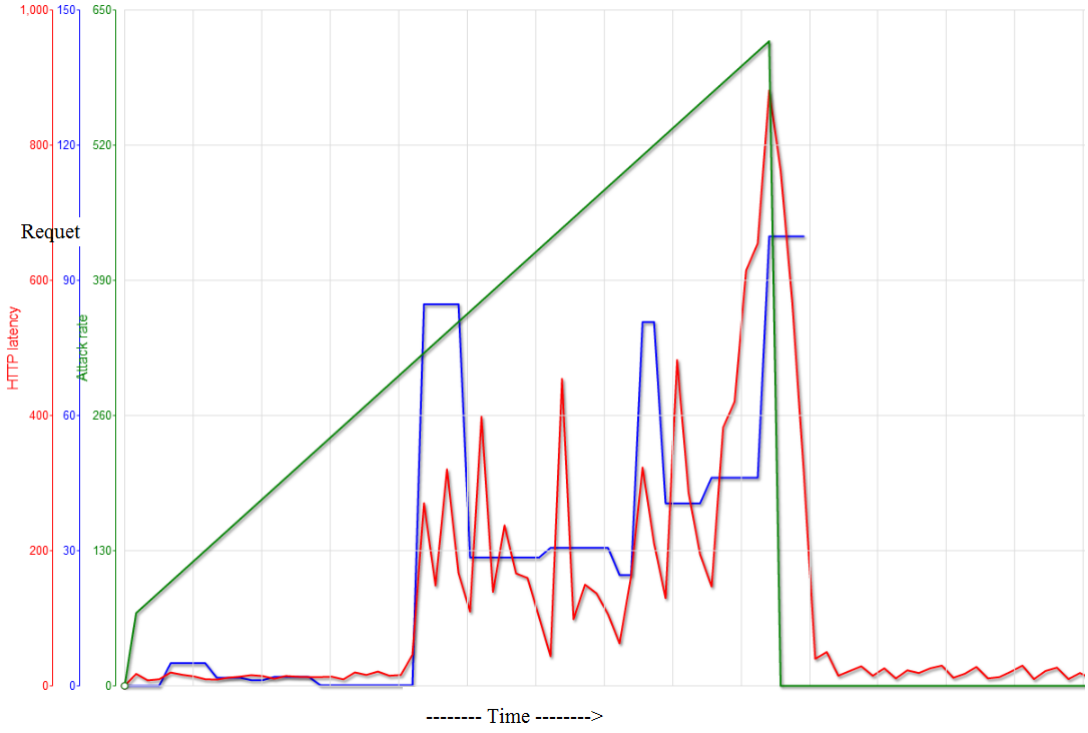

Figure 12: HTTP flood process

7 Conclusion

During this research we explored possibilities for doing controlled DDoS simulations. We analysed the observables of both the application- and network-level DDoS attack classes to find ways of monitoring these attacks. We found a number of different monitoring methods that unfortunately do not universally apply to both attack classes.

We chose to further test the packet-loss approximation for the network-level attacks and the slope prediction for application-level attacks, as described in section 5.3.2.

One of the key findings here is that relative low frequency probing is not very suitable for the monitoring of network-level attacks. Instead, it becomes a viable option to monitor the responses to a portion of the attack traffic, which we simulated with equally sized ICMP echo requests, to approximate the global packet-loss.

Application-level attacks in general may however show more gradual changes in observables, like the response time. We found that these values can be used fairly reliable to predict when a threshold is about to be reached.

Although more testing is required to test these monitoring methods on different attacks within their respective attack class, they show potential to reliably trigger on a threshold being reached. With the monitoring methods providing feedback, the proof-of-concept showed successful control over the attack simulations. For both attack classes we defined sample thresholds that were honored by the simulation, enabling DDoS testing without causing damage.

8 Future work

Based on the explored methods and findings of this research project we recognize a number of items that may require further attention to further develop on the concept of doing controlled DDoS simulations.

We believe the thresholds as discussed in section 4.1 are for a large portion user defined and highly dependent on the type of service. However, it may be the case that to some degree generalized values apply to the type of service or protocol in use. For instance, we found that roughly 1 percent of packet-loss on a 100Mbps link only just resulted in acceptable behaviour for HTTP traffic.

Another area of research could be to include the use of local monitoring instead of relying solely on the remote observables. We believe that non-out-of-band local monitoring would show low granularity during an attack simulation although there may be workarounds for this. Further research in this area could focus on the interpretation of local monitoring data and the communication from an attacked system to the master, for instance through out of band communication or resource reservations.

References

[1] S. Bhatia, D. Schmidt, G. Mohay, A. Tickle, *A framework for generating realistic trafc for Distributed Denial-of-Service attacks and Flash Events*, 2013
http://www.sciencedirect.com/science/article/pii/S0167404813001673

[2] S. Abu-Nimeh, S. Nair, M. Marchetti, *Avoiding Denial of Service via Stress Testing*, 2006
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1618370

[3] M. Yan, H. Sun, X. Wang, X. Liu, *WS-TaaS: A Testing as a Service Platform for Web Service Load Testing*, 2012
http://www.computer.org/csdl/proceedings/icpads/2012/4903/00/4903a456-abs.html

[4] T.V. Lakshman, U. Madhow *The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss*, 1997
http://www.kth.se/polopoly_fs/1.123418!/Menu/general/column-content/attachment/lakshman-tcp.pdf

[5] W. Su,C. Kung, T. Lin, C Wu. J. Hsu *An On-line DDoS Attack Traceback and Mitigation System Based on Network Performance Monitoring*, 2008
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4494041

[6] A. Kamali, M. Berkelaar *Controlled (D)DoS framework code repository*, 2014
https://github.com/mikeberkelaar/controlleddos

9 Appendices

9.1 Terms and Abbreviations

- DDoS: Distributed Denial of Service (attack)
- PPS: Packets Per Second
- RTT: Round trip time
- SNMP: Simple Network Management Protocol
- Master: Master is a role in our architecture which is considered to be as the orchestrator of the whole process. All decisions are made there and every Agent registers at and reports to the him.
- Agents: Agents are those processes in charge of doing the actual work. They register themselves at the Master and get their commands from him. They also send any probably feedback/report to the Master. There are two different Agents:
  - Attack Agent: An Attack Agent is in charge of performing attach via sending out the proper traffic with defined rate at defined moments. All the definitions are clarified by the Master.
  - Monitoring Agent: A monitoring Agent keeps its eyes on the target in order to gather its vital signs. It will then report back to the server with the raw data gathered from the target. Feedback variable: is the actual resource parameter being monitored as the vital sign of the target machine. Examples of such variable would be: CPU utilization, number of packets loss, Response time, and so on.
- Inoperability point (or point of inoperability) This is the point in which the service becomes unavailable and the DoS attack would completely take down the target. The ultimate goal of this project is to predict this point while being as close as possible so that we can tell for sure the service will become unavailable with next move. Traffic Types:
- Attack Traffic All the traffic an Attack Agent sends our to the target is considered as Attack Traffic. This traffic are supposed to cause the actual Denial of Service and henve share these common characteristics :
  - The sources address is spoofed;
  - They are being sent as fast as could without any delay (according to the defined rate of Master);
  - No responses is gathered regarding these traffic. Any probable feedback/response/answer is discarded as soon as possible.

– Monitoring Traffic: In contrast to the Attack Traffic, Monitoring Traffic is supposed to be as much of real traffic as possible. This traffic is supposed to simulate the behaviour of an end-user and work as an advocate of him to check (o defined intervals) if and how the service is still operating. Monirointg traffic is in a very low ratio compared to the attack traffic and onviously consists of legitimate source addresses.
– Global Traffic: The sum of all monitoring traffic being sent from Monitoring Agents and all attack traffic being sent from Attack Agents would be the Global Traffic which is the amount of traffic the target machine would be dealing with.

9.2 Code repository

The source code used as a proof-of-concept implementation is shared on GitHub at `https://github.com/mikeberkelaar/controlleddos`. [6]

9.3 Figures in Larger scale

Some of the figures in this documented are available with a higher resolution:

- Traffic flood: Figure 13
- HTTP flood: Figure 14
- Control panel: Figure 15

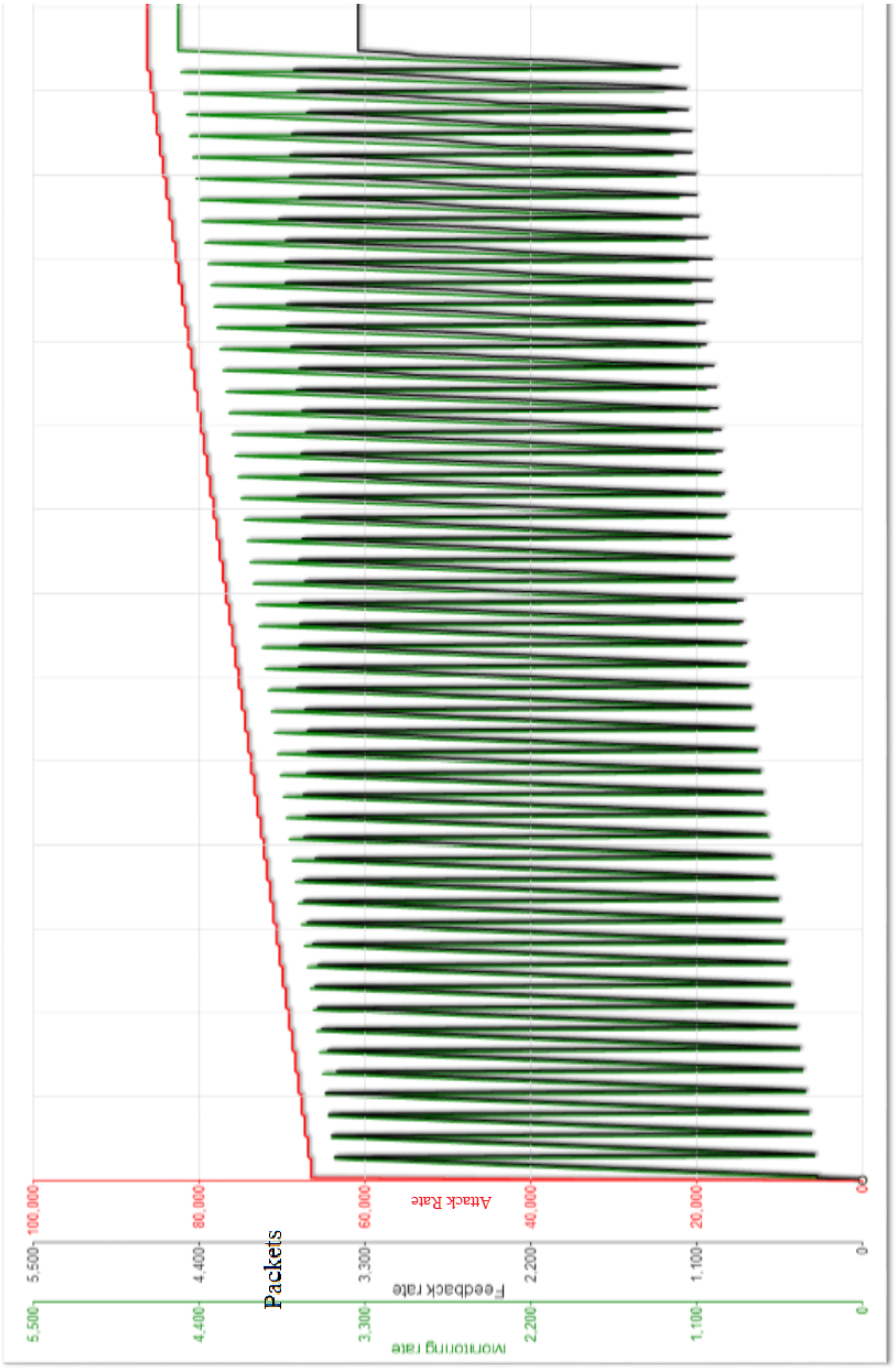

Figure 13: ICMP Flooding Attack Full

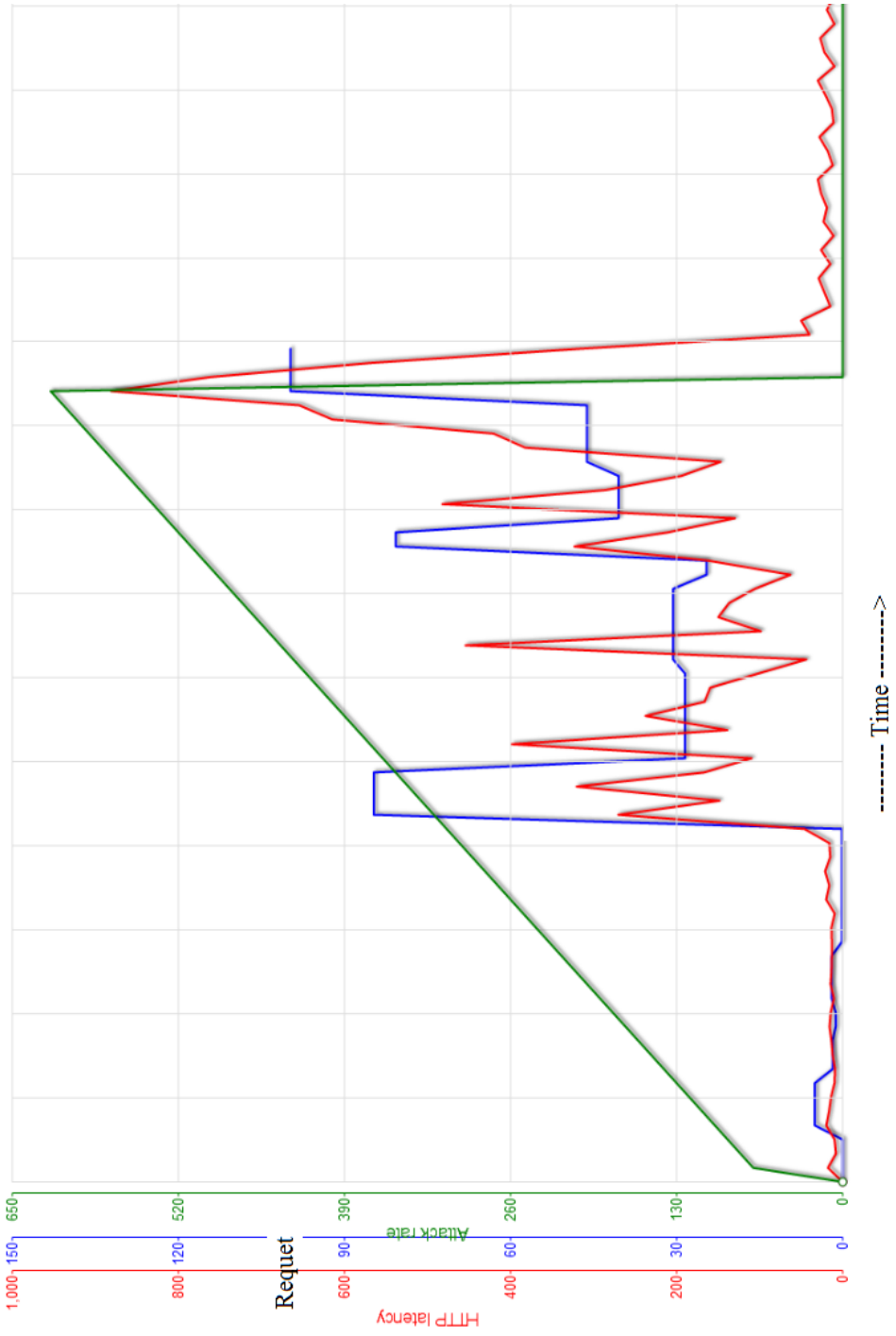

Figure 14: HTTP flood process

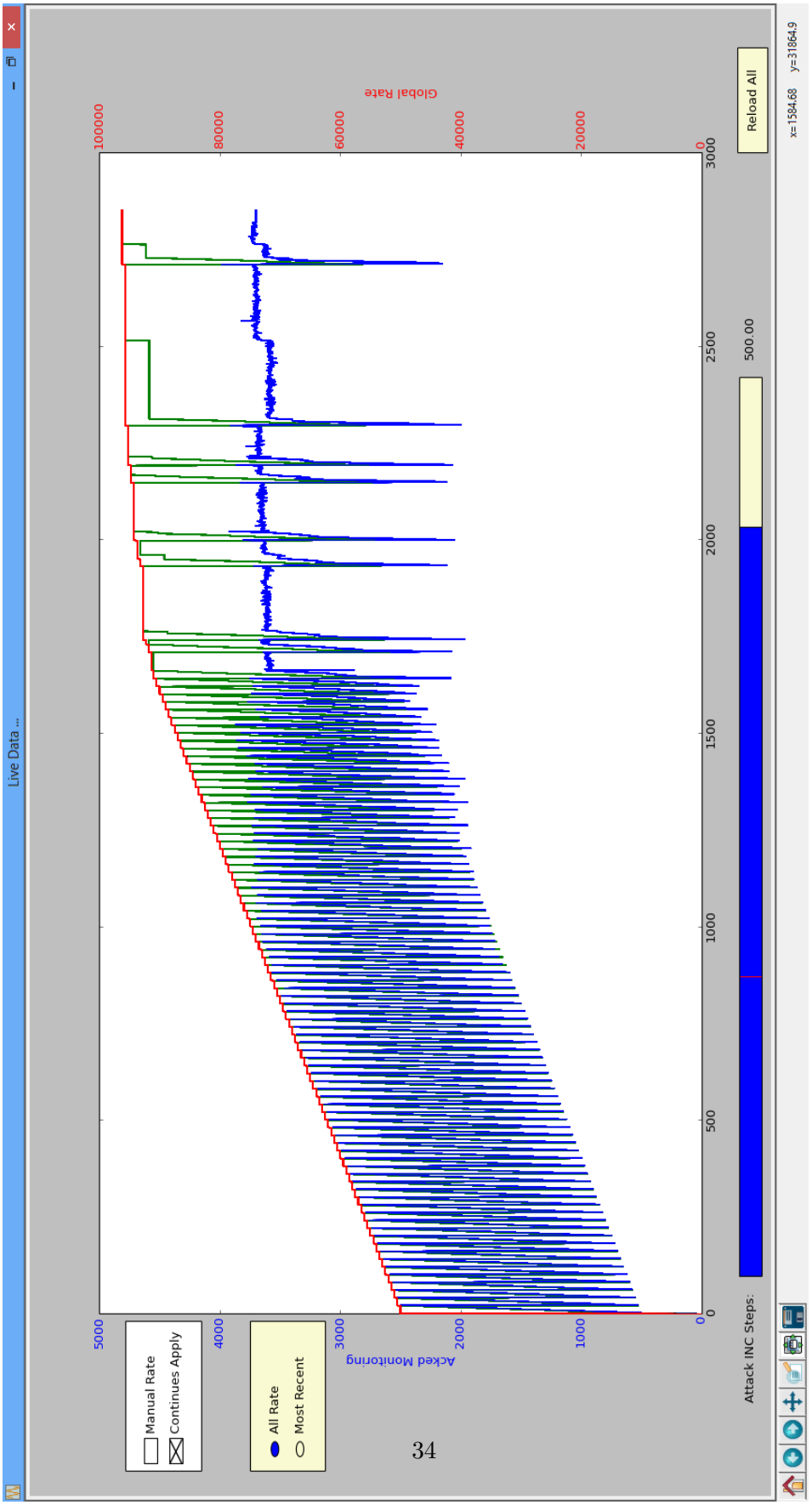

Figure 15: Master's Control Panel