Detecting BGP Interception Attacks using RTT Measurements

Daniel Jubas

ABSTRACT

Internet users run the risk of having their internet traffic surveilled potentially leading to leaked sensitive information or the loss of anonymity. Adversaries can manipulate the trust of the Border Gateway Protocol–the protocol which controls the routing decisions of the internet–to enable such surveillance by diverting victim traffic to their own servers. Existing methods of detecting such attacks are inefficient, too slow, or susceptible to being tricked by a clever adversary. We present here a novel method of detection based on the changes in latency of internet traffic. Efficient and fast methods for round trip time calculation in the data plane exist and because latency is bounded by topological distance it is difficult to spoof. We consider three latency-based detection algorithms for efficient online monitoring of traffic which can help real-time mitigation of ongoing attacks. We evaluate the algorithms' performance in detecting real (ethically executed) interception attacks and also assess its performance in avoiding false positives in real anonymized Princeton University internet traffic.

ACM Reference Format:
Daniel Jubas. 2021. Detecting BGP Interception Attacks using RTT Measurements. In *Proceedings of ACM Conference (SIGCOMM'18).* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

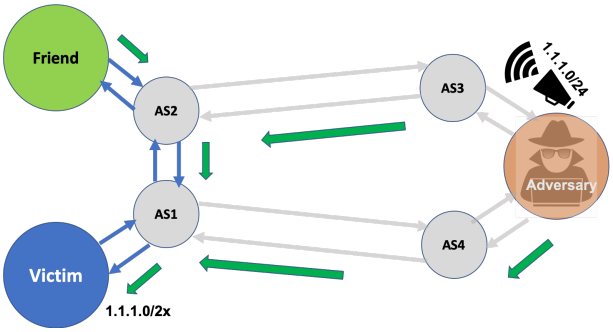
1 INTRODUCTION

By using the internet, users open themselves up to a wide array of attacks. One general form of attack is to eavesdrop on someone's communication with another internet endpoint. If the victims have neglected to encrypt their traffic, the eavesdropping adversary cna gain access to sensitive data such as passwords or private conversations. However, even if packet contents are encrypted, an eavesdropper can still extract sensitive information just by checking packet headers and seeing who is communicating with whom. Further, even users browsing on the Tor network–a software designed to provide anonymous internet use–or other similar services are still vulnerable to deanonymizing traffic analysis which can be accomplished if the adversary gains a large enough foothold into the network[1–4].

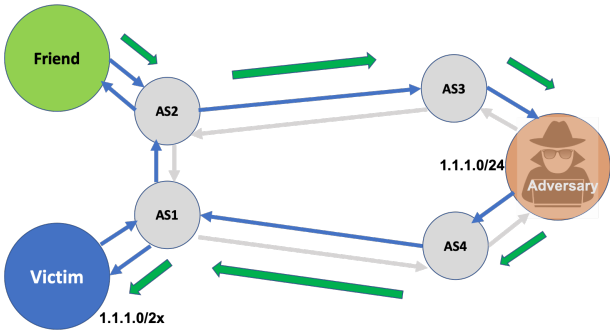
Such attacks can be made possible by the primary routing protocol for the Internet's backbone, The Border Gateway Protocol (BGP). This protocol does not have any means of secure authentication of Autonomous System (AS) identity nor the honesty of announced routes which allows a malicious AS to lie and broadcast a route it should not. Clever BGP misuse can allow for effective eavesdropping without the need to break into the victim's machine and does not require tapping into a router on the normal path of the victim's traffic. All an adversary needs to do is reroute the victim's traffic through its own servers via a bogus BGP announcement [5]. Such an adversary might simply monitor the traffic and drop it, but such a hijack attack would quickly be noticed by the victim as they no longer receive any communication. In a more subtle attack, known as an interception attack, the adversary diverts the traffic so it can be monitored but then forwards the traffic to the intended recipient [6]. In doing so, it makes it difficult for the user to realize anything is amiss (Figure 1). This makes interception attacks difficult to detect, especially in real-time.

This attack might have appeal to many nation-state actors for a variety of reasons. For example, a nation might be interested in tracking a political activist. While one could, for example, force a local dissident to have their traffic routed though government servers for close monitoring, we do not focus on that type of local case. Rather, we prioritize cases in which a nation state wants to intercept traffic overseas. For example, if this political activist has fled the country, or if the target is an official of an enemy state. Indeed, there are examples which suggest that this type of behavior might already be deployed. In 2011, government owned China Telecom hijacked 50,000 global prefixes diverting them to the China Telecom servers before most were forwarded back to their intended destination. Victims during this incident included US government agencies [7]. States alternatively might be interested in gaining access to foreign sensitive data as was one in in a 2017 Russian interception of US credit card traffic [8]. Another interesting use case places the US government in the role of adversary, circumventing a protection of citizens' internet communications from unwarranted search by diverting domestic traffic overseas where it no longer has these protections. Following major whistle blowing from Edward Snowden of the NSA, one could imagine this tool being used to try and monitor and suppress suspected political dissidents [9, 10].

Methods for detecting these BGP interception attacks do exist, but none are totally satisfactory. Some approaches leverage active probes such as traceroute [11]. However, such active methods are resource inefficient and make real



(a) Before interception attack



(b) After interception attack

Figure 1: Diagram of interception attack on a victim communicating with a friend. Green arrows represent BGP routing to the Victim. Blue arrows represent traffic between Friend and Victim. Grey arrows help show network topology (i.e. do not carry traffic between Friend and Victim but might carry other traffic). (a) The Victim and Friend are communicating with traffic traversing multiple ASes from source to destination. The victim has previously announced either a /23 or /24. The adversary, far away, announces a /24, equally specific or more specific than the victim's announcement, which propagates. (b) The announcement has converted AS2 and AS3 to forward traffic intended for the victim not to the best route to the victim, but to the adversary, who analyzes it before forwarding onward to the victim via AS4 which did not hear the bogus announcement.

deployment unfeasible. Other approaches focus on parsing propagated BGP announcements and checking for inconsistencies (such as conflicting origin ASes) [12] which can be inefficient. Additionally, it has been demonstrated that interceptions can be successfully executed without the BGP announcement ever reaching either of the effected endpoints making this type of approach difficult [13]. A complete review of related work can be found in Section 6.

One unexplored approach which resolves many of these issues is to use latency information to detect attacks. Round Trip Time (RTT) measurements involve recording the time a packet is sent and matching it with a reply acknowledgment (ACK) to calculate the latency. In this paper, we refer to a single latency calculation as a "sample." Abnormal round trip time, and in particular sustained increases in latency, could correlate with long range interception attacks as the traffic must traverse a greater topological distance. This approach has a number of beneficial qualities. Unlike other methods, this approach can be measured close to the source to avoid corrupting maneuvers from the adversary. That is, an overseas adversary cannot keep latency low while carrying out an interception attack since there is a lower bound set by topological distance. An RTT-based detection algorithm could in theory be used for *online* monitoring and respond in real-time to flagged behavior for instance by rerouting packets, making a more specific BGP announcement so that traffic is no longer diverted, adding in random delay (to prevent traffic analysis), or stopping traffic altogether depending on the sensitivity of the information.

The ability to quickly and efficiently monitor for interception attacks can be realized by an emerging but substantial technology in programmable switches. Programmable switches are highly controllable network devices which sit in the data plane and can passively measure RTT on long lasting TCP connections. While these switches offer speed and efficiency, they come with challenges such as low capacity memory and limited memory access abilities [14] requiring that any detection algorithm implemented on one of these switches be constrained to simple operations. We see incorporation into a programmable switch as a feasible direction towards deployment. Indeed, in order to be robust to all traffic analysis attacks, which can be completed with as few as 100 packets [3], the type of speed and efficiency offered may be necessary.

On top of these constraints, multiple other factors make developing a suitable algorithm for this task difficult. First, arbitrary factors relating to the hardware and use of the network lead to varying amounts of slowdown and noise which require differentiation from malicious attacks. For example, we used traces recorded on Princeton University's campus to evaluate the efficacy of the RTT-based approach and iterate on the design. While doing so we discovered traffic patterns which made this differentiation difficult, the cause of some we could easily explain and others we could not. Being effective in the face of the many diverse traffic patterns featured regularly in traffic in the wild all the while having the constraints of a programmable switch forces us to act as efficiently as possible. Second, the lack of available

datasets containing known interception attacks with its associated RTT data, in addition to ethical issues with running an interception attack on the open internet, makes testing and revising the algorithm challenging. Luckily the PEERING test bed [15], a series of prefixes and servers around the world for research use, allowed for the creation of real RTT data to help with the development.

In this paper, we implemented an RTT-based detection model in software and analyzed it in an ethical environment demonstrating high effectiveness at flagging true interception attacks and achieving fairly low rates of false positives. Ongoing work will transition to implementing it into a programmable switch, reducing the rate of false positives, and solidifying concrete mitigation steps.

The rest of the paper is structured as follows:

- Section 2 presents a detailed problem formulation.
- Section 3 introduces the algorithms considered for RTT-based detection.
- Section 4 describes the threat model we consider and explains the various interception attacks we used to test our detection approach.
- Section 5 evaluates the success of these algorithms in correctly detecting the interception attacks. Additionally, it analyzes parameter choices for the various algorithms by evaluating them on the Princeton traces.
- Sections 6 discusses current state of the field and why our approach better fits certain use cases than existing alternatives.

2 PROBLEM FORMULATION

In this section we further explain in what scenario RTT-based detection would be valuable.

2.1 A Motivating Example

Consider the motivating use case of n privacy-concerned user of the Tor network. Tor is a software which allows users to remain anonymous while browsing the web, making use of three intermediate servers between the communicating endpoints so that it is difficult to associate received traffic on one end with traffic being send on the others. Tor is widely used – it has over 7000 relays and about 4 million active users, and played a critical role in the Edward Snowden whistle blowing as well as the Arab Spring [16].

Not surprisingly, many entities have an interest in breaching the security Tor provides. Meanwhile, Tor, in order to avoid impractically high latency, makes concessions which open itself up to potential traffic-analysis attacks were an adversary to gain a sufficient foothold into the network. This could be achieved with a clever interception attack. Consider a Tor client using the web via the Tor network, passing through a number of ASes. An adversary may launch an interception attack in order to get on path of this traffic and run traffic analysis to deanonymize the victim and correlate its traffic with its destination! For the purposes of this paper, we assume that the adversary is sufficiently far off path to make a large enough increase (Figure 1).

2.2 Defense

Our novel insight is that the Tor entry or node can defend from these attacks by monitoring the latency of the traffic in real-time. RTT measurement is done by matching an outbound packet with its received acknowledgement (ACK) [17]. We can use these latency measurements as a form of proxy for topological distance in the Internet. Propagation delay is the amount of time it takes for a packet to reach its destination from a source. We consider round trip propagation which is the sum of the propagation delays of the packet and acknowledgment. In other words, the amount of time spent *in transit* to the exclusion of processing time at the far endpoint. This value sets a lower bound for the calculated latency as it does not include that processing time whereas the observed latency does. As the path the traffic takes changes increases, the propagation delay should increase accordingly and if processing time at the end is unchanged, we can expect observed latency to increase as well. Therefore, if we assume that the stable route BGP determines between two endpoints is close to optimal, an adversary far enough off-path who diverts this traffic will elongate the route and increase the latency. We can analyze the stream of latency data and flag any significant increases as this likely signifies a malicious routing change. Once there is a suspected attack, there can be a mitigation step or a suite of mitigation efforts that are taken to fight this attack. These mitigation efforts are beyond the scope of the paper but are touched on in brief in Section A.1.

2.3 Constraints

The problem we are solving is much more constrained than a regular change-point analysis problem. First, these attacks can be as quick. In 2018 [3] achieved 62% success at traffic analysis with only 100 packets, advances in the field since will surely bring this number down. In order to be a robust defense against such fast attacks, we must be able to respond that quickly. Therefore, this solution must be not only in real-time, taking a single pass through the data stream, but also decide quickly whether or not this pattern matches an attack. Second, because we see the most effective path towards deployment as integrating with programmable switches, we are constrained in the types of operations that we can perform on the switch. While programmable switches are a quickly developing field, we restrict ourselves to simple comparison and basic arithmetic operations. Additionally, due

to the speed constraints and the memory limitations of the switches, we only take a single pass through the latency data. Third, the solution must be widely generalizable. That is, if this is going to be deployed in a switch, at some endpoint or in a Tor relay, we cannot tailor sensitivity settings to each particular communication. Rather, we would like to come up with a solution which works well across any arbitrary two endpoints in communication. Fourth, this must be efficient in two senses: (1) it must be resource efficient in order to avoid large performance hits to latency and throughput of the network. Tor intentionally makes security sacrifices to prioritize a better performing browser experience. To offer a solution which simply reverses that intentional trade-off is effectively not offering a solution. (2) it must be accurate. While prefect accuracy is desired, we think it is impossible and therefore we need to consider the tension between having the detection be sensitive enough to flag true instances of an attack while not being overly sensitive to noise.

Amid all of those constraints, the problem still boils down to: given a stream of RTT samples for each client, can we detect large RTT changes within in fewer than 100 samples (and ideally under 50 samples) while avoiding high false-positives.

3 DETECTION ALGORITHMS

In this section we elaborate on three of the algorithms used for analysis. Additionally, we point out where some of them fell short, characterizing the type of communication which poses difficulty for accurate detection. In Section 5 we compare the performance of these algorithms and consider parameters which should be chosen for optimal detection.

3.1 Algorithm Background

If we consider a route between a source and destination to be relatively stable (i.e. all of the traffic is sent on the same route) we can think of the round trip time as the the combination of two components. One of these components is the round trip propagation delay. This number is theoretically a constant representing the amount of time that is required to travel from one endpoint to the other and back. The second component is variable and constantly changing even on a stable route. It is comprised of processing time at the destination as well as small delays caused by things such as buffering, queuing, reordering and resending packets during transit.

For stable connections, we expect this to be roughly the case and so round trip times hover above some theoretical stable line which is this round trip propagation delay. When a route experiences a change for benign or malicious reasons, the same principles will apply for the new route and there will be some new baseline lowest theoretical RTT.

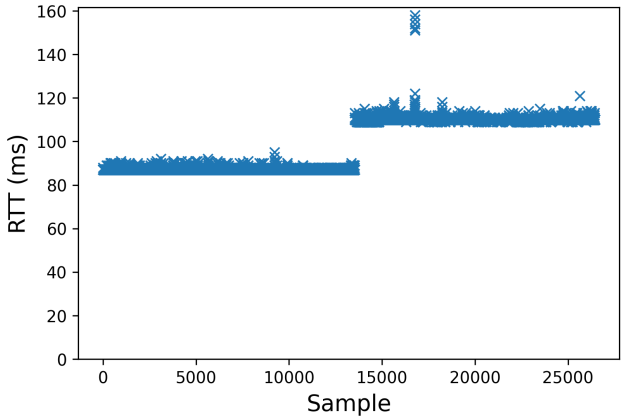


Figure 2: RTT versus Sample plot of an interception attack of communication between an AWS instance in Ireland and servers at Northeastern University near Boston, MA. Traffic is diverted to an adversary in Amsterdam. Each x represents a single RTT sample. Samples are in chronological order but evenly spaces and not plotted with time.

For example Figure 2 shows traffic stable around 90ms and then after the routing change it readjusts to be stable (albeit with more noise) above 115ms. In a straw-man case of zero noise any change would be detectable as the data is otherwise consistent (and there are indeed connections which exhibit roughly this behavior). Comparing adjacent packets for a large enough change reveals a routing change. Taking a slightly more realistic case with noise such as we have in this same figure, *despite* some noise, a large scale interception attack would be detectable with the same method so long as we only check for sufficiently large jumps.

Yet, individual samples might have lengthy RTTs due to something such as a delayed ACK. For instance, in Figure 3, delayed ACKs occur not infrequently at times around 100x of the primary signal. Less extreme events of this same phenomenon occur regularly – in an arbitrary chosen set of 100 consecutive flows (as arranged in the dataset) from a dataset of flows with more than 144 samples, 58 of the 100 had a delayed ACK where that is defined as singleton outlier at least 200ms greater than the regular RTT. Most of this 58 has outliers with latency thousands of miliseconds above the others. Accordingly, there is a need to view samples in aggregate and to characterize what the signal is from amidst the noisy data.

Specifically for this detection method, the signal is the RTT associated with topological distance. Because we have this lower bound from round trip propagation delay, the lower the round trip time, the closer it serves as a proxy for distance. Therefore, we focus on the lowest RTT values

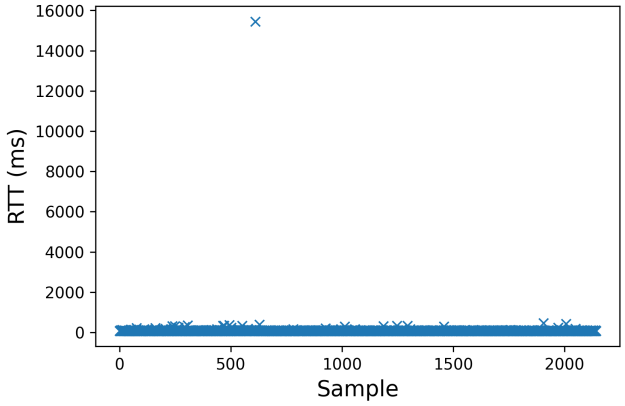


Figure 3: Example of delayed ACK at around sample 750. The RTT for that sample is close to 16 seconds which is about 100 times the bulk of the other samples.

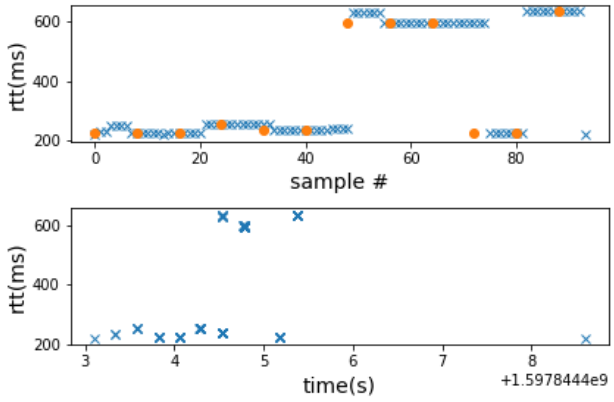


Figure 4: Two plots of the same connection. Top: Round trip time plotted versus the sample number with orange circles showing the minimums. This is how the algorithm 'sees' the data. Bottom: Round trip time plotted versus real-time (seconds since epoch).

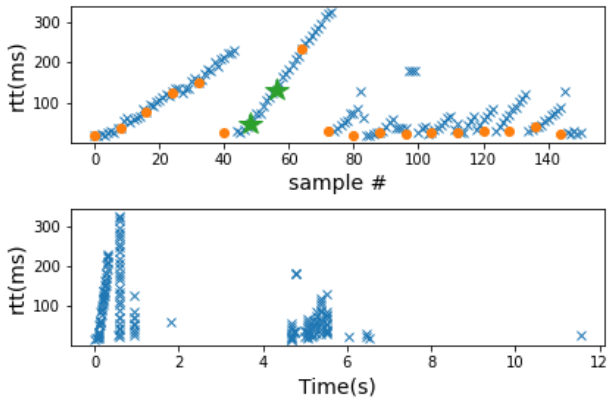


Figure 5: Top Buffer bloat image as seen from the perspective of a min-filtering detection algorithm. Blue 'x' mark data points, orange dots the minimum of each window, and green stars the points flagged by the two-window min filtering algorithm Bottom The same connection plotted versus time.

in the calculation of whether to flag an attack or not. We present a simple algorithm which leverages the power of the minimum to approximate distance while not being tripped up by outliers.

3.2 Two Window Min Filtering Algorithm

This algorithm breaks up the data into window sizes w and finds the minimum of each. Adjacent minimums are compared to see if there is a sufficient increase in round trip time, more than a threshold T, such that it is likely an adversary is intercepting traffic. Through min-filtering over a window of data, we both reduce susceptibility to noise and also only use the samples which best approximate topological distance. This algorithm also has the additional property of being easy to implement into code generally which makes it promising for implementation in the data plane. (See Section A.5).

3.3 Three Window Algorithm

The two-window algorithm works well in particular when traffic matches our initial expectations for what traffic might look like. However, it fails on noisier traffic and consistently on one pattern in particular: buffer bloat (Figure 5). Buffer bloat is the phenomenon of having overly large buffers in routers upstream of a lower bandwidth connection causing a steady increase in latency until the server sending traffic is informed to reduce the amount of traffic it sends, temporarily relieving the issue [18]. The resultant latency hike can be as large as a few hundred milliseconds. It has an identifiable shape as seen in Figure 5. Meanwhile, the two-window algorithm collapses a window of data into one data point for comparison. In buffer bloat, since we are using relatively small window sizes, each window captures part of the bloat, and when compared it appears that there is a jump. However, we really wish to be looking for a new *sustained* minimum RTT, not just a momentary increase.

To respond to buffer bloat, we extended the algorithm to three-windows which allows to look for *sustained* increase in latency to the exclusion of more temporary or volatile ones. To do so we define a third variable s for instability tolerance. Consider three windows w_1, w_2, w_3. In addition to a sufficiently large increase between the first two windows, it is also required that $abs(min(w_3) - min(w_2)) \leq s$ in order to flag this. With this addition, if the increase in latency is not

sustained (allowing for minimal deviance), it is presumed to be buffer bloat or noise of some kind. We considered values for s all below 10ms.

3.4 A Third Algorithm: Kth-min Filtering

A last algorithm was explored in response to a phenomenon among the campus traces in which there would be a stream of data, relatively steady around some latency and then a single *low* outlier. The way the algorithm this is very likely to trigger a flag in either of the other two algorithms. It is unclear what the cause of this issue is–likely it has to do with with peculiarities of the measurement as opposed to an actual short drastic decrease in latency. In order to explore this, we tried using a k-th min filtering scheme. Instead of the regular 0-th min filtering, we set k=1 in order to see if that could maintain the benefits of the standard min-filtering over a window, but slightly improve it by excluding these outliers.

4 EXPERIMENTAL SETUP

In order to test the efficacy of the detection algorithms, we executed interception attacks ethically in the wild using the PEERING test bed. PEERING runs geographically dispersed ASes which allow for researches to make real BGP announcements while not affecting other entities [15]. For the purposes of describing these attacks, we refer to three players. Prior to the attack, a *friend* and the *victim* are in communication over the internet. During the attack, an *adversary* diverts traffic intended for the victim (originating at the friend) to its own servers before forwarding it along to the victim.

4.1 Communities and Interception Attacks

In order for the interception attack to be considered successful, the adversary must divert traffic intended for the victim and then forward it onward to the victim. In a simple naive setup, this forwarding step is impossible. The traffic was diverted to the adversary because ASes along the way thought that this was the best route to reach the victim, if the adversary simply tries sending the traffic to the victim – the intended recipient– the intermediate ASes along the path will send it along the best route which is right back to the adversary. The traffic will never reach the victim and it will be obvious that something has happened effectively announcing the attack to the victim.

In order for the forwarding to actually succeed, it is crucial for there to remain enough ASes which actually believe that sending the traffic to the victim *not* via the adversary is the best route.

Communities are such a tool, which, although not intended for this purpose, can be utilized to achieve this end. Communities are attributes which can be attached to BGP announcements which provide for somewhat specific additional instructions allowing for more control over routing policy. Communities might be used to relay extra information about the announcement or to tell an upstream provider to do a certain action. Communities can also be used to tell providers to NOT propagate an announcement or to demote a local preference for a particular hop making the route, although propagated, not likely to be chosen. These latter two forms of community attributes can be used to make the bogus announcement from the adversary either entirely unknown to portions of the internet or known but unappealing to that portion (and effectively ignored). The main challenge of the adversary is then to keep the announcement unknown to part of the internet while still having the announcement propagate far enough to reach the victim and friend.

Using communities to enable an interception attack would still fail if there is only one provider to the adversary. Either that one provider thinks the best route is via the adversary in which case it will keep sending any received traffic back to the adversary, or it does not in which case it would not pass it to the adversary to begin with. Therefore, this attack requires the adversary AS to have two providers so that it can announce the victim's prefix to one of them and then forward it out of the one it does not announce to.

We assume that the adversary has access to at least two providers and can make announcements using communities so that it can run a successful interception attack. Using just BGP announcements, we consider three versions of an interception attack that an adversary could deploy: (1) an equally specific attack where the adversary gives an equally specific announcement as the victim, (2) a sub-prefix interception where the adversary gives a more specific prefix to increase attack reach, and (3) a stealthy sub-prefix attack where path poisoning is used to prevent the announcement from reaching both endpoints.

4.2 The Three Attacks

PEERING provides access to an mux in Amsterdam with two providers which is designated as the adversary for this reason–we can use one provider to announce the bogus announcement and the other to redirect traffic onward to the victim. We conducted three experiments using a variety of other locations. A PEERING mux in Northeastern University, in Cambridge, Massachusetts serves as the victim. In order to minimize added latency, these muxes are controlled using co-located AWS instances.

4.2.1 Equally Specific Interception Attack. Using an AWS instance in Northern Virginia, we controlled a PEERING mux at Northeastern University (NEU) near Boston, Massachusetts. We announced the PEERING-provided /24 prefix (e.g., 1.1.1.1/24) and ran a server there posing as an ip in

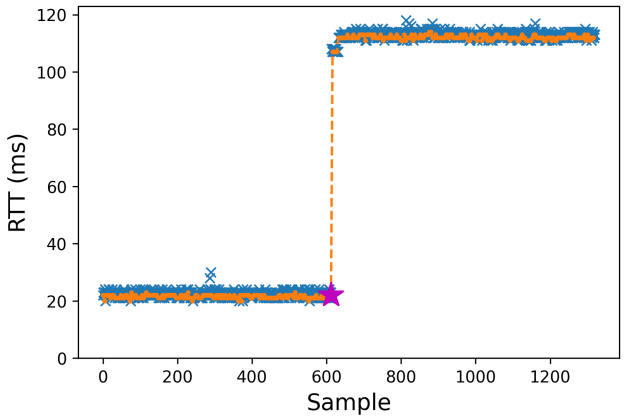


Figure 6: RTT versus Sample plot during an attack in which an AS in Amsterdam intercepts communication between Princeton and Northeastern Universities. Attack begins at around sample 600. Blue x represents single RTT sample, orange dots with dashed line is how the plot looks after min filtering and the star is where the detection algorithm things there is an attack beginning. All combinations of parameters tested with jump threshold less than 125 successfully located a singular attack in this flow.

that network. From an AWS instance in Ireland we set up a TCP connection with that ip-port pair and continuously sent packets to NEU which echoed back the same message. After 20 seconds of benign behavior, we launched the attack whereby the adversary in Amsterdam announces 1.1.1.1/24 with a number of community attributes and we then update Amsterdam's packet filtering rules to no longer respond to the prefix (as the owner) and instead forward the traffic via one of its providers to the victim. After another 30 seconds, we stopped recording. Packets and ACKs were recorded using tcpdump and tcptrace used in order to compile latency data.(exact commands can be found in the Appendix A.6).

4.2.2 Sub-prefix Attack. From NEU we announced the PEERING-provided /23 prefix (e.g., 1.1.1.1/23) and ran a server posing as an ip in that network and bound it to a particular port. From a machine in Princeton's network, we set up a TCP connection with an ip in the announced prefix space. After 20 seconds of benign behavior, we launched the attack whereby the adversary in Amsterdam announces 1.1.1.1/24 with a number of community attributes, updates its packet filtering rules to longer respond as the owner of that prefix and forwards the traffic via one of its providers to the victim. After another 30 seconds, we stopped recording. Packets and ACKs were recorded using tcpdump and tcptrace used in order to compile latency data.

This attack has a number of advantages and disadvantages compared to the equally specific interception attack. BGP paths are prioritized using a number of tiers. At the top tier, a more specific matching prefix always wins over a less specific one. After that there are a number of business-interest related considerations as a customer might not want its path to be known to a peer. Lastly, if routes are equal in the other areas, a shorter path length is preferred. Now, in order for an interception attack to work, it often means using many communities to avoid both of the adversary's providers from hearing the bogus announcement. Doing so often make the route from friend to adversary longer. As the path grows, it is less likely that an intermediate AS is going to pass along the bogus route, as it is increasingly likely that a different route is preferred due to being shorter. Therefore, there is a firmer limit on how far an equally specific attack is going to propagate. It is possible that everything will work out, however, this is very labor intensive to pull off with good reach. For instance, in an equally specific attack, adding in all of the communities required to make an interception attack work–that is, for one provider to announce the bogus path and it not be heard by the second– leads to the announcement not being heard at all at NEU or Princeton. Accordingly, the attack was run from Ireland. A sub-prefix attack, because prefix specificity is the primary metric for route acceptance, means if an AS does hear of the bogus announcement, it will be the preferred one and propagated on. This is why the attack was able to intercept traffic sent from Princeton even though Princeton is close to Northeastern and far from Amsterdam.

However, [6] already makes the suggestion of only announcing /24 in order to avoid the ease of a sub-prefix attack. An equally specific attack is then less preventable and perhaps therefore more likely to be deployed in the wild by a real adversary. We demonstrate its success and that it can be detected by the three algorithms.

4.2.3 Stealthy sub-prefix attack. From Seattle, we announced the PEERING provided /23 prefix and ran a server from there posing as an ip in that network and bound it to a particular port. From a PEERING mux in Wisconsin controlled by an AWS instance in Ohio we opened up TCP connection with a PEERING mux in Seattle controlled by an AWS instance in Oregon. After 20 seconds of benign behavior, we launched the attack from Amsterdam. This attack features, in addition to the added communities, a path-poisoning of the last hop before the Friend (the mux in Wisconsin). We updated Amsterdam's packet filtering rules to no longer respond to the prefix (as the owner) and forward the traffic via one of its providers to the victim. After another 30 seconds, we stopped recording. Packets and ACKs were recorded using tcpdump and tcptrace is used in order to compile latency data.

This attack is very similar to other subprefix attack. However, the BGP announcement does not propagate to the victim nor the friend. Therefore, any passive detection scheme relying on seeing BGP announcements at the victim or friend is not going to be successful. However, for an RTT-based approach, we can nonetheless detect this attack.

5 EVALUATION

In this section we evaluate the efficacy of the three algorithms and do parameter tuning to determine at which settings are the algorithms best. The high-level goal of this section is motivated by the desire for the detection schemes to be deployable. To be successful in a real setting, a detection algorithm must be tuned to be sensitive enough so that it flags attacks when they do appear while at the same time not being too sensitive such that short-lived perturbations are not often mistaken for an attack and flagged. We consider both of these metrics by both running them against actual interception attacks and also by running it on benign (presumed not-intercepted) traffic. In both settings we consider the three algorithms with the following settings: window size of 8, 16, 24, 32, 48; jump threshold of 5, 10, 20, 30, 40, 50, 75, 100, 125, 200ms; and instability tolerance of 2 and 9ms.

5.1 High-Level Parameter Discussion

The approach we took for recommendation of parameters was to use the high quantity of campus data and see which parameters would be unlikely to falsely raise a flag while still being sensitive the real attacks. That is, we want to have as few false negatives as possible while keeping false positives low. Here we briefly explain some of the considerations.

5.1.1 window size. The size of the window affects the granularity with which the detection looks for changes. Since for each window we have just one data point, too large a window size leads to potentially missing quick attacks. For instance, if the window size is 64 samples, this means that it requires 128 samples to extract two windows in order to detect a change and 192 for three. An efficient attacker could do traffic analysis within that amount of time and withdraw the BGP announcement, causing the attack to be missed by the detection–a false negative. At the same time, too small a window size leads to over sensitivity to what is just very noisy data. A small window size would lead to false positives when what appears to be an attack is just a passing phenomenon. We therefore need a window size which is not too small as to mistake fleeting noise with an attack and is not too big that it is too slow to be useful or mistakes a real attack for fleeting noise.

5.1.2 jump threshold. This parameter T represents a threshold value. If an $i+1$-th window is more than T greater than the i-th the two-window algorithm will flag it as an attack (and the three-window will if it sustains this heightened latency through the next window). When selecting an appropriate value for T there are two primary considerations to make. The first is that the larger T the further off-route the adversary must be in order for the attack to be captured. For instance, in the NEU-Ireland-Amsterdam attack, a jump value of 30ms is already too great to detect this attack which increases RTT by about 25ms (Figure 2). The second is that if T is too small noise will more frequently be mistaken for an attack (e.g., Figure 9). In order for the detection scheme to remain fairly simple so that it can be deployed in a programmable switch, we think we must select a single parameter for detection and so the value chosen becomes a trade off between sensitivity to noise and accuracy in selecting only true positives. We make a recommendation, however ultimately different scenarios will call for different weightings of costs and benefits–a government owned switch monitoring top secret information would likely be more tolerant of false-positives than a general purpose switch run by an ISP or a Tor node.

5.1.3 instability tolerance. This parameter, s is only relevant in the three-window algorithms. In those, we wish to see that the change is a *sustained* one. If the $i+1$-th window deviates from the i-th window by more than s in any direction, then we suspect that the spike was due to noise and not an attack and do not flag it. As s increases, more jumps in RTT will remain with that limit and be flagged as an attack. s must be low enough to avoid over flagging noise but high enough to allow for the possibility of an attack to take place with some instability.

Two methods were used to determine the efficacy. We considered: window sizes of 8, 16, 24, 32 and 48; jump values of 5, 10, 20, 30, 40, 50, 75, 100, 125, and 200ms; and instability tolerances of 2 and 9ms. The goal is to find a set or ranges of parameters which we determine to have a balance between false positives and false negatives. That is, being sensitive enough to detect attacks from close range and quickly while also limiting the amount of noise which mistakenly flags attacks. The PEERING experiments are used in order to see whether or not chosen parameters are able to detect the attacks when they do occur–avoiding false negatives. Meanwhile, we make use of a dataset of 35 million connections from a day of Princeton University campus internet traffic, in which we assume there are no interception attacks, to see how the parameters do at avoiding false positives.

5.2 PEERING Results

In total 26 attacks were run. Generally, all of the algorithms performed well at detecting the attacks. Compared to some of the other traces studied, the PEERING experiments involving

NEU had relatively little noise (Figure 6). In the absence of noise, as expected, all three of the algorithms performed well. Two of the attacks were unsuccessful to begin with and therefore none successfully flagged it.

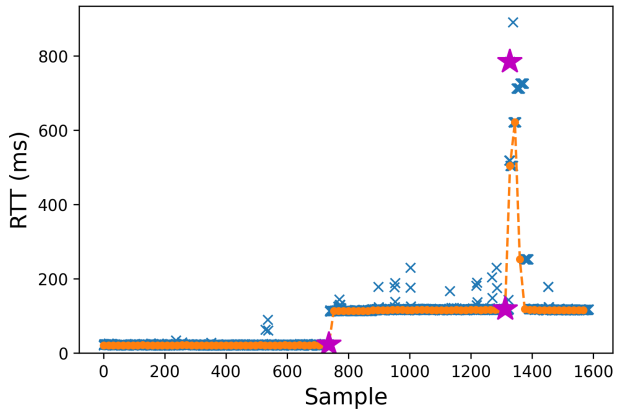
As expected, the 200ms threshold did not catch any of the attacks correctly. The size of RTT increase in the sub-prefix attack on NEU-Princeton traffic often had a latency increase of around 80-100ms so the 125 and most of the time 100ms attack were unable to make detection. The Seattle-Wisconsin interception had more variable increases in latency, at times well above 100ms at times slightly below. But in order to be able to detect all of the successful attacks, a jump threshold of 20ms would be required as the equally specific interception only diverts that much (Figure 2). That being said, in one instance, due to what seems to be slow to converge to the new route, a jump threshold of 10ms would have been required (Figure 15).

There were no instances, when, with the same window size and jump threshold, that the three-window algorithms with an instability tolerance of 9ms were unable to detect the attack but the two-window algorithm was able to. The instances when there was noise were more informative for differentiating the different algorithms. The two window algorithm did not fair well in the presence of noise. To take one example, in Figure 7 which the performance of the two- and three-window algorithms, the two-window algorithm, because it only compares adjacent windows, mistakes a steep incline in latency due to noise for an attack.

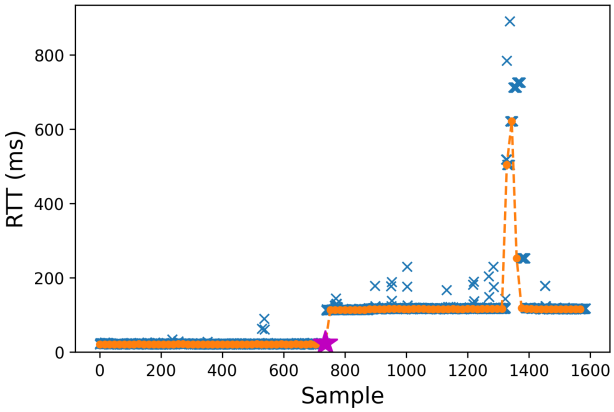
Similarly, the window sizes also were differentiated in the presence of noise. Figure 8 compares two parameter settings, all the same but in one the window size is 8 and in the other 16. As can be seen there, while both of them have register the noise in their minimum. Because window size of one is larger, there is only one minimum that is elevated and the detection algorithm distinguishes that as noise while with a smaller window size, that is considered to be a sustained change. Setting the window size to 32 or greater entirely smooths out the curve, ignoring all of noise around sample 550.

Instability tolerance was in most instances, not a deciding factor between flagging the attack or not. However, in 3 of the attacks the difference between 9ms and 2ms instability tolerance made the difference between flagging and not flagging.

Similarly, the window sizes also were differentiated in the presence of noise. Figure 8 compares two parameter settings, all the same but in one the window size is 8 and in the other 16. As can be seen there, while both of them have register the noise in their minimum. Because window size of one is larger, there is only one minimum that is elevated and the three window algorithm differentiates that as noise while with a smaller window size, that is considered to be a sustained



(a) Two-window algorithm assessing an interception attack. It locates three attacks when there should be one. $w = 16T = 75ms$



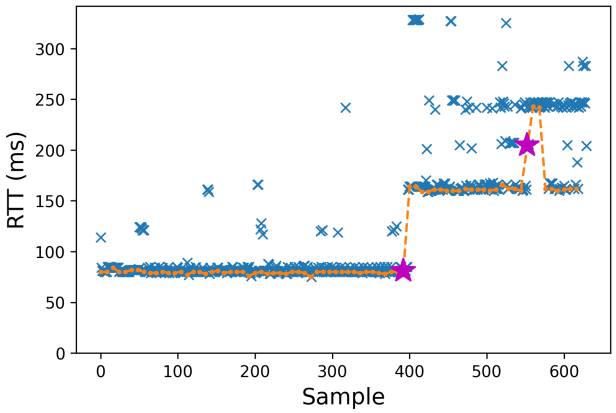
(b) Three-window algorithm assessing an interception attack. It locates just one attack as it should be. $w = 16T = 75ms, s = 9ms$

Figure 7: Plots of performance of the two and three window algorithm on an attack diverting traffic between Princeton and Northeastern Universities to an adversary in Amsterdam. Blue x is a single RTT sample, the orange dots and dashed lined represent the min-filtered data and the purple stars are locations of suspected attacks.

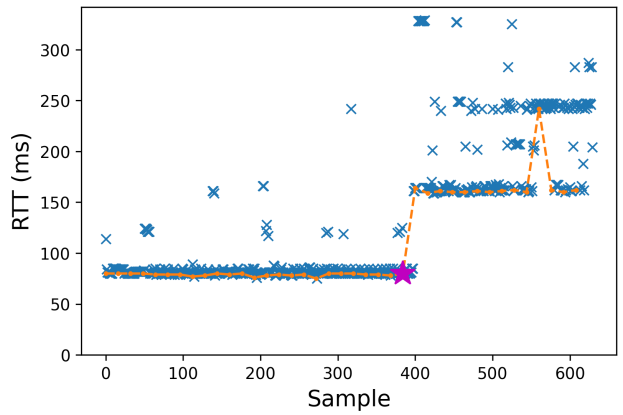
change. Setting the window size to 32 or greater entirely smooths out the curve, ignore all of noise around sample 1350.

Instability tolerance was in most instances, not a deciding factor between flagging the attack or not. However, in 3 of the attacks the difference between 9ms and 2ms instability tolerance made the difference between flagging and not flagging.

There are five takeaways from the PEERING experiments: (1) affirms that these attacks are each able to be carried out



(a) Three window algorithm set with $w = 8$, $T = 20$, $s = 2$. Shows two suspected attacks.



(b) Three window algorithm set with $w = 16$, $T = 20$, $s = 2$. Shows one suspected attack which is when the attack did occur.

Figure 8: RTT versus Sample plot for an interception attack diverting communication between PEERING mux in Seattle and Wisconsin to a Amsterdam. (a) has a window size of 8. (b) has a window size of 16. Traffic may be being routed along two paths during the attack leading to this noise. Blue x is a single RTT sample, the orange dots and dashed lined represent the min-filtered data and the purple stars are locations of suspected attacks.

on the internet today, (2) that this RTT-based detection approach, in particular either of the three-window algorithms, can be used to successfully detect these attacks, (3) some attacks require being very sensitive and setting jump threshold as low as 20ms or even 10ms, (4) Having a larger tolerance for instability can make the difference between detection and not even in relatively low noise flows, and (5) smaller window sizes do lead to fewer false positives but at least from here do not make it more effective at locating the attacks themselves.

5.3 Campus Data Results

We analyzed traces taken from a day of Internet traffic recorded at Princeton University and anonymized in accordance with IRB standards for approval. The original dataset contained 35,000,000 flows of which we created two subsets. We assumed that there were no actual interception attacks in the dataset. The goal is to select parameters which are impervious to much of the noise in all of these connections while not being so conservative as to not raise any flags.

5.3.1 Methodological Considerations . In working with this campus data set, a number of methodology decisions were made. First, from the large dataset, two smaller ones were created. One contained all flows with more than 96 packets (244,541 flows) and one all flows containing more than 144 packets (165,522 flows). These were used for the two and three-window algorithm analysis respectively. This was because we wanted to test the algorithms on the same flows across window sizes. Since we consider a window size of 48 we only used data which could accommodate that large a window size. It should be noted that were the other option taken – and datasets on which large window sizes could not be run were used – it would make the larger window sizes look comparatively better than they do as it would add in a number of flows they would certainly not flag.

We also considered what metric to use in order to compare the different parameter tunings. One option is to see how many flows from the dataset are flagged. On the one hand, this approach does not give undo weight to some select very noisy flow which has what appears to contain many attacks. On the other hand, it also does not factor in that some flows are much longer than others–a single flagged instance in a flow of 100 samples is different than one in a flow of 100,000 samples. Another approach is to look at flags per windows in a flow. This normalizes for opportunities to flag across different window sizes. This approach ignores the fact that the larger-window sizes do simply have fewer opportunities to raise flags. A last approach was to look at the rate of flagging per *sample*. The main issue of too many false positives is that there will be some effort involved in mitigating this suspected attack which should not be wasted. The metric which best relates how often those inefficiencies will impact performance is done by considering suspected attacks per sample. We made use of all three although felt that suspected attacks per sample was the most relevant.

This opens up another question: what should the target rate of suspected attacks per sample be at which we are satisfied with the algorithms performance. Because the actual mitigation steps are their associated costs are not known,

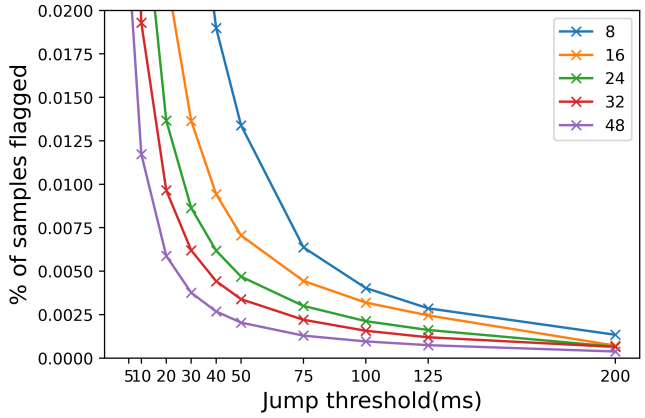
it is hard to come up with a number with any real level of confidence. It could be that having even 50 false positives in a day is a decent performance hit. However, it could also be the case that mitigation is a light process and thousands of false positives would not go noticed by the user. This does indeed leave a large range of acceptable values. We ultimately decided on the target of one false positive per 100,000 samples. While the original dataset had 35,000,000 flows, the overwhelming majority had fewer than 5 samples (with 0 having by far the most). That is, while this number can be extrapolated to the thousands of false positives per day at Princeton's campus, if efficient mitigation steps can be put in place, this number would hopefully not come with a significant performance hit. A discussion of ideas on potential mitigation steps can be found in Section A.1.

5.3.2 Two-Window Results. Looking to Figure 9, the results for jump thresholds up to 200ms are plotted. As can be seen from the graph, as the threshold increases, there are fewer suspected attacks. This makes sense as the events which get a flag at a high threshold are a subset of what gets a flag at a lower one. Additionally, in line with expectations, as window size increases there are fewer flagged events. However, looking to part b of the same figure, it seems that this might be attributed to simply having fewer instances to flag and not a qualitative difference in the algorithms.

To achieve the 1 in 100,000 goal would require a threshold of no less than 75ms and to keep the algorithm quick (fewer than 50 samples required) would need closer to 175ms. Returning to the PEERING data, a jump threshold would not be reliable for most interception attacks. The two-window approach in its current state would not be economical to deploy.

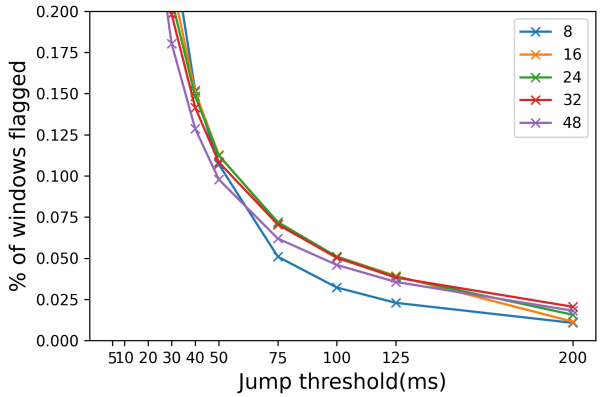
5.3.3 Three Window Results. The three-window algorithm outperformed the two-window algorithm quite conclusively. Comparing Figures 9a and 10a it can be seen that for a fixed window size and jump value, the three-window algorithm has a lower rate of flagging connections and windows by about an order of magnitude. It could have been the case that because the two-window algorithm is faster (in that it only takes two-windows to detect an attack and not three) that there might be scenarios when it would be preferred due to speed. However, because the three-window algorithm seems to be so much less likely to flag an interval, it would be preferred to choose a smaller window size with the three-window algorithm such that it is as fast as the two-window algorithm, rather than use the two-window with a larger window size.

If we stick to the 1 flag per 100,000 samples goal, we can achieve that with thresholds as low as 20ms (with window of 48 and instability tolerance of 2ms). If we wish to keep detection under 50 samples, we can select, for example, window



(a) Percentage of samples flagged versus jump threshold. Shown for various window sizes. Window size and jump threshold are both inversely proportional with the number of suspected attacks.

.



(b) Percentage of *windows* flagged versus jump threshold and plotted for various window sizes. Analysis was still run using dataset of only flows with more than 96 samples.

Figure 9: Two-Window Parameter Evaluation.

size of 16, threshold of 40 and instability tolerance of 2ms. While this is not sensitive enough to detect the neu-ireland-amsterdam attack, with a jump threshold of 40ms, it can detect many attacks–certainly more than the two-window algorithm. Alternatively, tolerating some more false positives (1 in 40,000) we could indeed pick a threshold of 20ms and catch even that attack. Finally, these numbers to some extent work for the worst case–an adversary who can accomplish their goal extremely quickly. Allowing for 100 samples which the overwhelming majority of attacks still cannot use effectively, we can use window size of 32 and either be able to detect the Ireland attack with a †threshold of 20ms or cut false positives manifold with a threshold of 50ms.

Figure 10b shows a plot of the three-window algorithm with various window sizes and with two values for the instability constant, 2 and 9. In this plot, we normalize over window size so that we can see less of an impact from window size. Of note, compared to Figure 9b which shows a similar plot but for the two window algorithm, the different window sizes differentiate themselves more with the three window algorithm. Additionally, we see the effect of a great instability tolerance as that those with a value of 9ms basically are all flagging more flows than any of the 2ms versions.

5.3.4 Kth-min Results. This last algorithm fared very similarly to the regular three-window algorithm. As can be seen in Figure 11, there is no noticeable difference between the regular three window algorithm (k=0) and the k-th min algorithm with k=1. In fact, looking closely, it seems that the k=1 algorithm is more likely to flag these non-intercepted flows.

5.4 Evaluation Summary

The regular three-window algorithm seems to be the most effective algorithm. It performs much better than the two-window algorithm, making (presumed) false positives at about $\frac{1}{10}$ the rate as the two window algorithm. It is very comparable to the k-th min algorithm when k = 1. Because the k-th min algorithm requires slightly more memory and offers no clear benefits, we recommend implementing the regular three-window algorithm. For parameters, the main trade off appears to be the window size and the jump threshold. A larger window size comes with fewer false positives, but means that the detection happens less quickly. That being said, a larger window size allows for selection of a lower jump threshold which allows for the detection to be usable on a wider range of attacks. Targeting a rate of 1 flag per 100,000 samples, if we wish to keep the window size small we can use a window size of 16 with a threshold of 40. This, as we saw will not be able to detect the attack on NEU-Amsterdam communications. Another possibility is to select window size of 32 with a threshold of 20ms. This allows for the detection of many more attacks, but comes at the risk of an increased delay to detection, putting it very close to the 100 packet amount that [3] needed to deanonymize with 62% accuracy. There are likely even faster attacks in development to this day. If fast detection is not a large concern (as the majority of traffic analysis requires more time [3]) then it could even be considered to use a threshold of 40ms with a window size of 32 to achieve a false positive rate closer to 1 in 500,000.

6 RELATED WORK

There are two distinct aspects to related work in this project. The first relates to the goal of detecting BGP network attacks, and more specifically interception attacks. The second relates to the method of detection and in particular to the field of changepoint analysis–finding when meaningful changes occur within a noisy signal.

6.1 Previous Work Detecting BGP Attacks

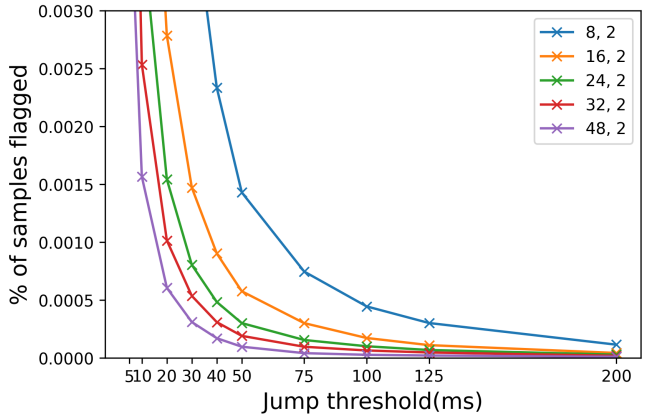
Here we give a short review of some of the approaches others have taken in tackling this problem and explain why an RTT-based detection is still worth detecting.

6.1.1 Active Probing. One approach to checking for BGP interception attacks is to actively probe the internet and check for some kind of peculiarity which is correlated with an attack. In [11] the AS being monitored uses persistent traceroute monitoring of 3000+ ASes to detect for prefix hijacking. Their method of detecting, based on noting when pings do not return, only works to signify an attack if the traffic is dropped but not an interception attack. Additionally, this method does a complete trace of the ASes roughly every half-hour which leaves much room for an undetected quick attack–Tor traffic analysis has been demonstrated to effective in just one minute [2]. [19] is another example of an active probing approach. [20] also uses active probing but does so from many monitoring points in order to look for path disagreements which are indicative of an attack. The distributed nature of this procedure requires having access to many dispersed monitoring points across the internet, making it not easily deployable. Additionally, the method checks for interception attacks to a particular prefix roughly every 12 minutes. In contrast, our detection scheme provides actual real-time monitoring so that there are no holes in monitoring.

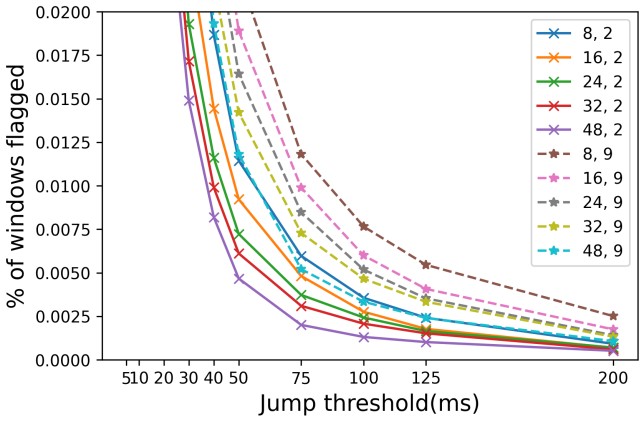
6.1.2 Detection in the Control Plane. Another approach taken is to look for signs of an attack in the control plane. [12] for instance uses real-time monitoring of BGP announcements to quickly spot attacks. Similarly, [21–24] also utilize available BGP information for their analysis. However, as is demonstrated by the stealthy sub-prefix attack (Section 4.2.3) information from BGP announcements can not be relied on necessarily.

6.1.3 Proactive Measures. A complimentary approach to the monitoring taking place in the data and control planes is to try and find suspicious ASes before they have launched an attack to begin with. [25] used a set of known serial hijackers and machine learning to try and distill behavior that is indicative of serial hijackers in order to classify others ASes as such. This is an interesting approach although could does not actually do real-time monitoring so it does not help detect an attack if the adversary is not part of a blocked AS.

6.1.4 Changes to BGP. Perhaps the most effective of any solution is to tackle interception attacks at their roots: the



(a) Percentage of samples flagged by the three-window algorithm versus jump threshold. Plotted for various window sizes and an instability tolerance of 2ms.



(b) Plot of windows flagged as a percentage of total windows versus jump size. Plotted for multiple window sizes and two instability tolerances.

Figure 10: Three-Window Parameter Evaluation.

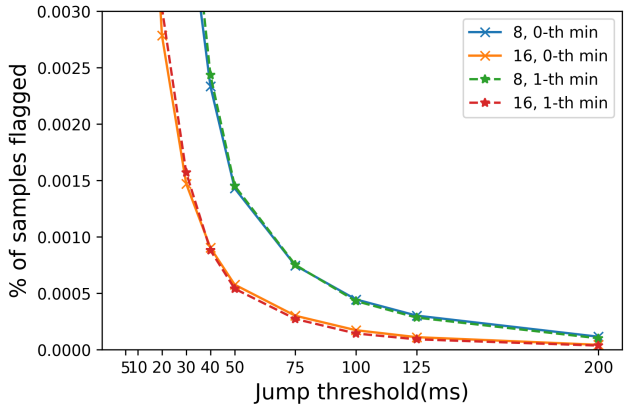


Figure 11: Plot of three-window algorithm using a kth-min compared to that of regular three-window ($k = 0$). As can be seen there is negligible if any difference between the two algorithms. Looking to the data point of window size 8, jump threshold 40, and window size 16, jump threshold 30, it can be seen that the regular algorithm slightly outperforms the other.

vulnerabilities of BGP. Updating the insecure protocol to allow for secure authentication would prevent attacks from being able to happen in the first place. It gets rid of the need for a detection algorithm at all. The most famous of such proposals is s-BGP, a secure version of BGP which makes use of public key infrastructure and digital signatures to enable secure authentication [26]. Similar approaches such as [27] and [28] trade some security for other benefits. All of these offer important fundamental solutions to the vulnerabilities of BGP. However, these papers are 15 to 20+ years old and an overhaul of BGP has not occurred. In the meantime, it will be necessary to have effective and robust attack detection such as the one offered by our detection scheme.

6.2 Change-Point Analysis

The attack detection falls under the broad category of change-point analysis – we wish to detect when there is a significant change in the signal amidst noise. In this particular case it is a change in the rtt signal. This is an area of on going study. For example, the Ruptures [29] provides a usable Python library for determining when these change-points occur. It is intentionally easy to use and is able to detect multiple change-points within a signal. However, ruptures fails to be usable for our task as it does offline analysis. [30] runs an online changepoint detection algorithm but it (along with [29] seems to assume that there are changes to detect, whereas in our use case, we do likely have no changes to detect. [31] implemented a Bayesian approach to changepoint detection in a microprocessor, but this approach would likely not be easily feasible on a programmable switch yet. In sum there are a lot of changepoint analysis techniques, all of which are more sophisticated than what we made use of here, however none of them currently can well serve our needs of being fast, online, and simple to implement. Future efforts especially with more advanced switches might be able to utilize these complex approaches.

7 CONCLUSION

Adversaries on the internet pose serious threats to user privacy and anonymity. Vulnerabilities in the Border Gateway Protocol and advances in research and technology now make it possible to deanonymize users in under 100 packets [3] which necessitates a fast online algorithm to help detect and respond to these attacks before they can complete their malicious tasks. We have demonstrated the potential for RTT measurements to form the basis of such a detection scheme, demonstrating its efficacy on ethical interception attacks and on regular web traffic. While the concept has been demonstrated, future work will need to implement this scheme into a programmable switch and likely some effort will be made or trade-off taken to get the false positive rate even lower. This paper serves as a good first step toward a practical and sufficiently effective defense against these attacks.

8 ACKNOWLEDGMENTS

There are many people to thank for getting this thesis across the finish line. First and foremost I would like to thank my advisor Professor Jen Rexford for giving me this opportunity to research in her lab and giving guidance and advice along the way. I'd also like to thank Joon Kim and Sata Sengupta for their assistance in helping me on board into the world of network security and for giving quality input into the project and its direction. I also want thank Sata in particular for his willingness to meet with me even outside of our weekly meeting despite it sometimes being the middle of the night for him.

I also owe a large debt of gratitude to Henry Birge Lee for all of the assistance he lent to a fellow senior out of the kindness of his heart. Getting these interception attacks up and running would likely not have been possible without his help.

I want to thank Professor Prateek Mittal for reviewing my presentation, offering feedback, and agreeing to be my second reader for the thesis.

Thank you to Ethan Katz-Basset and Italo Cunha over at PEERING for maintaining the test-bed and allowing me to use it for this project and for answering my multiple questions and requests.

Thank you to the internet and in particular all of the strangers who post and comment on stackoverflow and tex.stackexchange. Without you all, I would be sifting through documentation every time I needed to sort an array and bashing my head against the wall trying to align my figures in Microsoft Word.

Thank you to my friends and family for simultaneously expressing both support and also deep concern for my decision to opt-in to a thesis.

REFERENCES

[1] A. Mitseva, A. Panchenko, F. Lanze, M. Henze, K. Wehrle, and T. Engel, "Poster: Fingerprinting tor hidden services," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1766–1768, 2016.

[2] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal, "{RAPTOR}: Routing attacks on privacy in tor," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pp. 271–286, 2015.

[3] M. Nasr, A. Bahramali, and A. Houmansadr, "Deepcorr: Strong flow correlation attacks on tor using deep learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1962–1976, 2018.

[4] A. Bahramali, R. Soltani, A. Houmansadr, D. Goeckel, and D. Towsley, "Practical traffic analysis attacks on secure messaging applications," *arXiv preprint arXiv:2005.00508*, 2020.

[5] O. Nordström and C. Dovrolis, "Beware of bgp attacks," *SIGCOMM Comput. Commun. Rev.*, vol. 34, p. 1–8, Apr. 2004.

[6] H. Birge-Lee, L. Wang, J. Rexford, and P. Mittal, "Sico: Surgical interception attacks by manipulating bgp communities," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 431–448, 2019.

[7] R. Hiran, N. Carlsson, and P. Gill, "Characterizing large-scale routing anomalies: A case study of the china telecom incident," *Passive and Active Measurement*, 2013.

[8] D. Goodin and A. Smack, "Russian-controlled telecom hijacks financial services' internet traffic," Apr 2017.

[9] A. Arnbak and S. Goldberg, "Loopholes for circumventing the constitution: Unrestrained bulk surveillance on americans by collecting network traffic abroad," *MICH. TELECOMM. TECH.*, January 2015.

[10] S. Goldberg, C. Taylor, Berkowitz, J. Taylor, S. Miles, and A. E. Rogers, "Surveillance without borders: The "traffic shaping" loophole and why it matters," June 2019.

[11] Z. Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, and R. Bush, "ispy: Detecting ip prefix hijacking on my own," in *Proceedings of the ACM SIGCOMM 2008 conference on Data Communication*, pp. 327–338, 2008.

[12] P. Sermpezis, V. Kotronis, P. Gigis, X. Dimitropoulos, D. Cicalese, A. King, and A. Dainotti, "Artemis: Neutralizing bgp hijacking within a minute," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2471–2486, 2018.

[13] "Closed presentation with henry birge-lee." personal communication, March 2021.

[14] X. Chen, H. Kim, J. M. Aman, W. Chang, M. Lee, and J. Rexford, "Measuring tcp round-trip time in the data plane," in *ACM SIGCOMM Workshop on Secure Programmable Network Infrastructure*, (New York, NY, USA), August 2020.

[15] B. Schlinker, T. Arnold, I. Cunha, and E. Katz-Bassett, "Peering: Virtualizing bgp at the edge for research," in *ACM SIGCOMM Conference on Emerging Networking Experiments And Technologies*, pp. 51–67, 2019.

[16] "Servers."

[17] X. Chen, S. Landau-Feibish, M. Braverman, and J. Rexford, "Beaucoup: Answering many network traffic queries, one memory update at a time," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pp. 226–239, 2020.

[18] M. Allman, "Comments on bufferbloat," *SIGCOMM Comput. Commun. Rev.*, vol. 43, p. 30–37, Jan. 2012.

[19] X. Hu and Z. M. Mao, "Accurate real-time identification of ip prefix hijacking," in *2007 IEEE Symposium on Security and Privacy (SP'07)*, pp. 3–17, IEEE, 2007.

[20] C. Zheng, L. Ji, D. Pei, J. Wang, and P. Francis, "A light-weight distributed scheme for detecting ip prefix hijacks in real-time," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 277–288, 2007.
[21] H. H. Alshamrani, *Detecting IP prefix hijack events using BGP activity and AS connectivity analysis.* PhD thesis, University of Plymouth, 2017.
[22] Y.-J. Chi, R. Oliveira, and L. Zhang, "Cyclops: the as-level connectivity observatory," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 5, pp. 5–16, 2008.
[23] J. Wu, Z. Morley Mao, J. Rexford, and J. Wang, "Finding a needle in a haystack: Pinpointing significant bgp routing changes in an ip network," *nsdi*, 2005.
[24] X. Shi, Y. Xiang, Z. Wang, X. Yin, and J. Wu, "Detecting prefix hijackings in the internet with argus," in *Proceedings of the 2012 Internet Measurement Conference*, pp. 15–28, 2012.
[25] C. Testart, P. Richter, A. King, A. Dainotti, and D. Clark, "Profiling bgp serial hijackers: Capturing persistent misbehavior in the global routing table," in *Proceedings of the Internet Measurement Conference*, IMC '19, (New York, NY, USA), p. 420–434, Association for Computing Machinery, 2019.
[26] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (s-bgp)," *IEEE Journal on Selected areas in Communications*, vol. 18, no. 4, pp. 582–592, 2000.
[27] P. v. Oorschot, T. Wan, and E. Kranakis, "On interdomain routing security and pretty secure bgp (psbgp)," *ACM Trans. Inf. Syst. Secur.*, vol. 10, p. 11–es, July 2007.
[28] J. Karlin, S. Forrest, and J. Rexford, "Pretty good bgp: Improving bgp by cautiously adopting routes," in *Proceedings of the 2006 IEEE International Conference on Network Protocols*, pp. 290–299, IEEE, 2006.
[29] C. Truong, L. Oudre, and N. Vayatis, "Selective review of offline change point detection methods," *Signal Processing*, vol. 167, p. 107299, 2020.
[30] W. Shao, J.-L. Rougier, A. Paris, F. Devienne, and M. Viste, "One-to-one matching of rtt and path changes," in *2017 29th International Teletraffic Congress (ITC 29)*, vol. 1, pp. 196–204, IEEE, 2017.
[31] T. Figliolia and A. G. Andreou, "An fpga multiprocessor architecture for bayesian online change point detection using stochastic computation," *Microprocessors and Microsystems*, vol. 74, p. 102968, 2020.

A APPENDIX

A.1 Potential Mitigation Steps

A large under explored question from this project is what the effort to actually mitigate an interception attack look like once one is suspected of being underway.

A.1.1 Stop sending intercepted traffic. This is perhaps the most drastic response we might take. In order to avoid revealing very sensitive information traffic is simply stopped. Perhaps after a some number of seconds packets are sent out to see if the latency has gone back down, signifying the attack is over or it was only a passing flourish of noise

A.1.2 Actively probe this route. Upon suspecting that traffic is being diverted to an adversary, it can then make sense to try and investigate the matter further. For example traceroute could be used here to see if the route being taken is surprising (e.g., two endpoints in the continental US have their traffic routed through Europe). It is true that there are methods the adversary can use to trick traceroute into not knowing that it is on path. In particular, traceroute works by sending out a series of packets with the Time To Live (TTL) set beginning at zero and incremented with each subsequent probe sent out. The TTL expires when it reaches the next hop and a message to that effect is sent back to the sender which is used to know what AS is at each step. When a clever adversary received a probe which should expire, it can simply increment the TTL and pass it along so it avoids detection However, for these purposes, it might be sufficient to note that the route is going well out of its way even if we are not confident that we are getting the full picture. This step can done as a way of figuring out if it is safe to begin sending traffic again, or, traffic could simply be slowed until discovering the results of the active probe.

A.1.3 Announce a more specific prefix. Some of the other methods suggested such as stopping traffic or probing the route for more information are potentially important steps but do not actively work to dislodge the attack. As discussed in Section 4.2.2, BGP routing decisions are made most decisively by considering the specificity of a prefix. A more specific prefix will be chosen over a less specific one whether or not one is longer than the other. Therefore, in theory to respond to an adversary's attack, the victim could make a more specific announcement so that routers will prefer this route over the adversary's.

A.1.4 Timing Variations. If the main concern of the attack is traffic analysis, one approach to mitigate that attack is to make it more difficult to correlate the traffic. This can be achieved by adding random delays to sending traffic through the network and relaying it to the next node. While Tor specifically does not do this for the sake of efficiency, perhaps if there is a suspected attack, it might make more economical sense to add that mechanism in.

A.1.5 Use different Tor relays. The onion routing system which Tor utilizes, uses three different relays in between the source and destination. Traffic analysis often works by gaining access to traffic of particular nodes which give needed coverage of the communication. But Tor generally does not need to use any particular set of relays. Therefore, for mitigation in the Tor network, selecting a new Tor relay, or perhaps changing all three might be sufficient to avoid the interception attack. While this line of thinking would be interesting to explore, one of the risks associated with the approach is that the detection would not be able to detect if this new route is already in the midst of an attack. The detection assumes at some point the route was not intercepted and then looks for an increase in latency. If the adversary is staging a large scale attack on Tor and the new relays are also affected, it will not be noticed by the detection.

A.2 Reducing the false positivity rate

One way to reduce the false positives is to improve the detection algorithm. Perhaps via a more clever design or using of more complex operations. A different way to effectively reduce the false positives is to be able to ignore some of them. One idea which follows this line of thinking is to keep track of which flows have recently been labeled by the algorithm as being intercepted. If, within a particular time frame there are enough suspected attacks involving the same location, it is much more likely that there is an attack on that prefix than if there is just one. Potentially different responses could be taken based off of the confidence that this is an attack and not noise. This type of approach allows for a higher tolerance of false positives as little is necessarily done each time a theoretical attack is flagged. This same mechanism could also be used to help assess whether or not the switch is at the victim end or the friend end. If there is just a single communication then there is no differentiation–both see heightened RTTs. However, if the switch is seeing many different flows then it would expect to see many if not all of those experience heightened RTTs. If it is just the friend, seeing only one flow experience higher latency is to be expected. This might have implications for the form that mitigation takes. For example, announcing a more specific prefix only has the hope of working if it is your prefix which has been hijacked. Announcing your own prefix when you are the friend will not stop the adversary in any form. What you might want to do is alert the actual victim (potentially using a different medium of communication).

A.3 Characterizing Some Traffic Patterns

While looking through many plots of the Princeton University data, there were a few observations made which, although not entirely relevant to the paper, are worth documenting and are done so here.

A.3.1 Delayed ACK. Delayed ACKs occur in the dataset with great regularity, with more than half of an arbitrary 100 flows having a delayed ACK. While usually there are just a sample or two with extremely high latency, occasionally there are many (Figure 12).

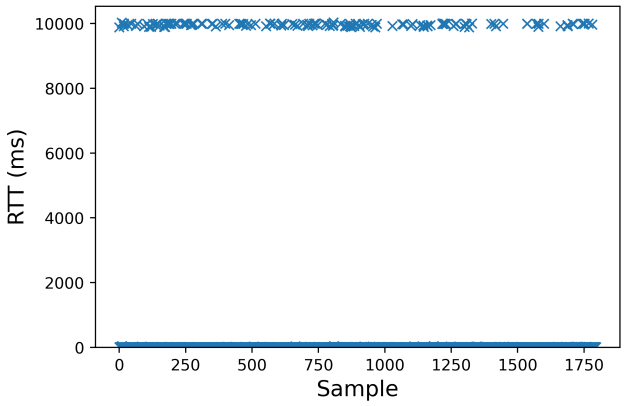


Figure 12: Plot of flow from campus data. This flow has 1796 packets and as can be seen a significant number of them are large outliers relative to the main bulk of samples at the bottom of the plot

A.3.2 Cellular Connection. One pattern which consistently gives very volatile RTT data are cellular data connections. These post challenges for the detection algorithms and at the most conservative tunings of the three-window algorithms, a high percentage of the false positives are in traffic which looks Figure 13 (and often is verified to be coming from an ip associated with a cellular company). The plot shown is a small fraction of the roughly 800,000 sample connection lasting about forty minutes. The external ip here is 71.168.128.73 which is attributed to MCI Communications an acquired telecommunications company. This is the first 2000 packets indicative of the whole which, if plotted, is hard to make much of due to the density. While often cell data looks like this, we do not assert that a cellular connection is the only cause of such patterns.

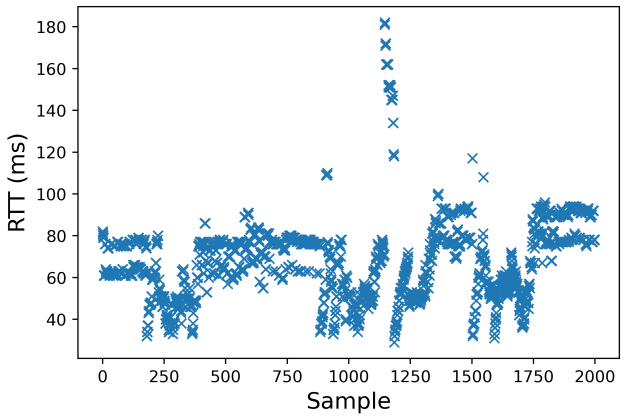


Figure 13: Partial plot of campus trace in connection with presumed cellular device.

A.3.3 Potentially Bifurcated Routes. There were some plots which were hard to make sense of. A rather significant number of flows exhibited the behavior seen in Figure 14 which makes it seem as though there are two distinct paths from source to destination and some packets take some and some take others. In fact on an observation of 100 arbitrary flows with more than 96 packets, 41 out of 100 had, to varying degrees of clarity, this behavior of two distinct lines being visible.

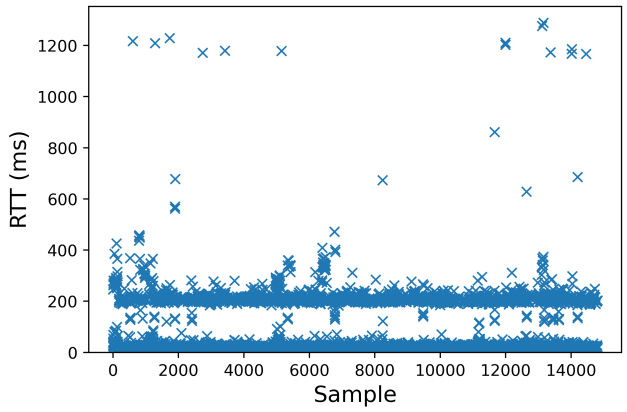


Figure 14: Plot of potentially dual-path plot.

A.4 Extra PEERING plots

A.4.1 Slow convergence. In this instance, the route did not immediately switch, but instead change in two small jumps of about 10ms each. n

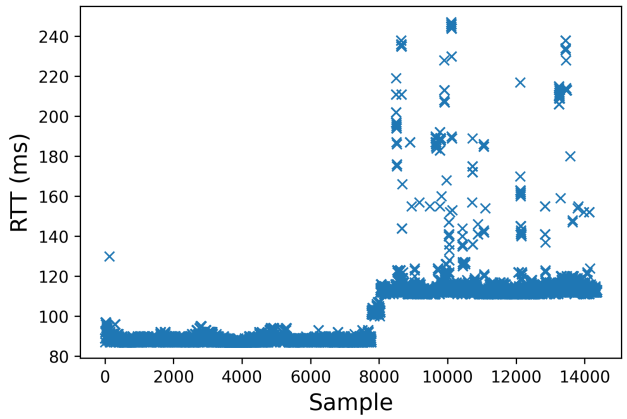


Figure 15: Plot of RTT versus Sample for an interception of NEU-Ireland communication where traffic was diverted via Amsterdam. A blue x is an RTT sample. The route was slow to converge and so only jump thresholds of 10 and 5 were able to detect it.

A.5 Algorithm Code

A.5.1 Two Window. Psuedo-code for implementing two-window algorithm in streaming fashion for online monitoring.

```
def twoWindows(stream, windowSize, jumpThreshold):
    oldMin = POSITIVE_INFINITY
    while True:
        newMin = POSITIVE_INFINITY
        for i in range(windowSize):
            sample = stream.ReadInt();
            if sample < newMin:
                newMin = sample
        if newMin - oldMin > jumpThreshold:
            raiseFlag()
        oldMin = newMin
```

Listing 1: Pseudocode for implementation of two-window algorithm.

A.5.2 Three Window algorithm pseudocode. .

```
def threeWindows(stream, windowSize, jumpThreshold,
     instabilityTolerance):
    firstMin = POSITIVE_INFINITY
    secondMin = POSITIVE_INFINITY
    thirdMin = POSITIVE_INFINITY
    while True:
        newMin = POSITIVE_INFINITY
        for i in range(windowSize):
            sample = stream.ReadInt();
            if sample < newMin:
                newMin = sample
        thirdMin = newMin
        if secondMin - firstMin > jumpThreshold and
             abs(thirdMin - secondMin) <
             instabilityTolerance:
            raiseFlag()
        firstMin = secondMin
        secondMin = thirdMin
```

Listing 2: Pseudocode for implementation of three-window algorithm

A.5.3 kth-min. This is how to implement the kth-min algorithm. The design avoids sorting.

```
def threeWindowsKthMin(stream, windowSize,
     jumpThreshold, instabilityTolerance, k):
    mins = []
    for i < k:
        mins[i] = POSITIVE_INFINITY
    while True:
        newKthMin = POSITIVE_INFINITY
        kthMinIndex = k-1
```

```
for i in range(windowSize):
    sample = stream.ReadInt();
    if sample < newKthMin:
        mins[kthMinIndex] = sample
        newKthMin = -1;
        for j = 0; j < k; j++:
            if mins[j] > newKthMin:
                newKthMin = mins[j]
                kthMinIndex = j

thirdMin = newKthMin
if secondMin - firstMin > jumpThreshold and
     abs(thirdMin - secondMin) <
     instabilityTolerance:
    raiseFlag()
firstMin = secondMin
secondMin = thirdMin
```

Listing 3: Pseudocode for implementation of kth-min algorithm

A.6 PEERING attack commands

Once PEERING account is made and you have a mux up and running (see their documentation) you can easily run an attack. Announce your prefix from the victim (should be a peering mux e.g., NEU). Then pick an ip and from provided prefix and from the mux run a simple TCP server script posing as that ip and listening for connections. From a different computer/server be the TCP client and continuously send messages back and forth. You can use the following for monitoring traffic at the friend.

```
python2 client.py &
sudo tcpdump 'dst 1.1.1.x and port yyyy and tcp' or
    'src 1.1.1.4 and tcp and port yyyy' -w
    tcpdumpoutput.pcap &
# wait
sleep(20)
echo "run the attack!"
sleep(30)
sudo pkill tcpdump
sudo pkill python
tcptrace -nZ --output_dir="tcptraceOutputs"
    tcpdumpoutput.pcap > summary.txt
```

Listing 4: Commands for monitoring experiment from the friend. Should have server running at the victim before running these commands.

Once that is running have the victim be the server, listening for a connection and the other endpoint the client.

A.6.1 Launching equally specific or sub-prefix attack on NEU. From the PEERING mux in Amsterdam run these commands when ready for the attack. These communities work for communication between the PEERING NEU mux and an AWS instance in Ireland for a equally specific attack (i.e., NEU announces a /24) or can be with NEU the victim and a much wide range of locations for the friend (e.g., Princeton) if this is a sub prefix attack (i.g., NEU announces a /23). Replace 1.1.1.0 with the PEERING provided prefix.

```
./peering prefix announce -m amsterdam01 -c
    47065,1061 -c 47065,1060 -c 0,12859 -c
    65520,2203 -l 8283,4,12859 -l 8283,4,34307 -l
    8283,4,286 -l 8283,4,15703 -l 8283,302,0 -l
    8283,4,2914 1.1.1.0/24
ip addr del 1.1.1.1/24 dev lo
ip route add 1.1.1.0/24 via 100.69.0.61
```

Listing 5: Commands for running attack on NEU from Amsterdam

A.6.2 Launching the stealthy sub-prefix attack. If the attack is being launched against Seattle, here is the stealthy sub=prefix attack [13]. AS 3128 is the poisoned AS.

Listing 6: Commands for running a stealthy attack on Seattle from Amsterdam

```
./peering prefix announce -m amsterdam01 -c
    47065,52 -l 8283,301,0 -l 8283,303,0 -l
    8283,4,15703 -l 8283,4,38930 -l 8283,4,57866 -l
    8283,4,6777 -l 8283,4,34307 -c 8455,5510 -c
    8455,5519 -c 8455,5523 -c 8455,5000 -c
    1299,2009 -c 1299,5009 -c 1299,7009 -p 3128
    1.1.1.0/24
ip addr del 1.1.1.1/24 dev lo
ip route add 1.1.1.0/24 via 100.69.0.61
```