Detection of Selfish Nodes in Networks using RED Algorithm

Kumbam Vekatreddy
Assistant professor, Department of Information Technology
Vignana Bharathi Institute of Technology, Hyderabad.

***Abstract*— In this paper, we consider the problem of detecting whether a compromised router is maliciously manipulating its stream of packets. In particular, we are concerned with a simple yet effective attack in which a router selectively drops packets destined for some victim. Unfortunately, it is quite challenging to attribute a missing packet to a malicious action because normal network congestion can produce the same effect. Modern networks routinely drop packets when the load temporarily exceeds their buffering capacities. Previous detection protocols have tried to address this problem with a user -defined threshold too many dropped packets imply malicious intent. However, this heuristic is fundamentally unsound; setting this threshold is, at best, an art and will certainly create unnecessary false positives or mask highly focused attacks. We have designed, developed, and implemented a compromised router detection protocol that dynamically infers, based on measured traffic rates and buffer sizes, the number of congestive packet losses that will occur. Once the ambiguity from congestion is removed, subsequent packet losses can be attributed to malicious actions.**

Keywords*— *Internet dependability, Intrusion detection and tolerance, Reliable networks, Malicious routers.

I. INTRODUCTION

The Internet is not a safe place. Unsecured hosts can expect to be compromised within minutes of connecting to the Internet and even well-protected hosts may be crippled with denial-of-service attacks. However, while such threats to host systems are widely understood, it is less well appreciated that the network infrastructure itself is subject to constant attack as well. Indeed, through combinations of social engineering and weak passwords, attackers have seized control over thousands of Internet routers. Even more troubling is Mike Lynn‘s controversial presentation at the 2005 Black Hat Briefings, which demonstrated how Cisco routers can be compromised via simple software vulnerabilities. Once a router has been compromised in such a fashion, an attacker may interpose on the traffic stream and manipulate it maliciously to attack others-selectively dropping, modifying, or rerouting packets. Several researchers have developed distributed protocols to detect such traffic manipulations, typically by validating that traffic transmitted by one router is received

unmodified by another. However, all of these schemes-including our own-struggle in interpreting the absence of traffic. While a packet that has been modified in transit represents clear evidence of tampering, a missing packet is inherently ambiguous: it may have been explicitly blocked by a compromised router or it may have been dropped benignly due to network congestion.

In fact, modern routers routinely drop packets due to bursts in traffic that exceed their buffering capacities, and the widely used Transmission Control Protocol (TCP) is designed to cause such losses as part of its normal congestion control behavior. Thus, existing traffic validation systems must inevitably produce false positives for benign events and/or produce false negatives by failing to report real malicious packet dropping. In this paper, we develop a compromised router detection protocol that dynamically infers the precise number of congestive packet losses that will occur. Once the congestion ambiguity is removed, subsequent packet losses can be safely attributed to malicious actions. We believe our protocol is the first to automatically predict congestion in a systematic manner and that it is necessary for making any such network fault detection practical. In the remainder of this paper, we briefly survey the related background material, evaluate options for inferring congestion, and then present the assumptions, specification, and a formal description of a protocol that achieves these goals. We have evaluated our protocol in a small experimental network and demonstrate that it is capable of accurately resolving extremely small and fine-grained attacks.

II. LITERATURE SURVEY

A Content Delivery Network or Content Distribution Network (CDN) is a system of computers networked together across the Internet that cooperate transparently to distribute content for the purposes of improving performance and scalability. Content types include web objects, downloadable objects (media files, software, and documents), applications, real time media streams, and other components of internet delivery (DNS, routes, and database queries). Strategically placed edge servers decrease the load on interconnects, public peers, private peers and backbones, freeing up capacity and lowering delivery costs. It uses the same principle as above. Instead of loading all traffic on a backbone or peer link, a CDN can offload these by redirecting traffic to edge servers. CDNs deliver content over TCP and UDP connections. TCP throughput over a network is impacted by both latency and packet loss. In order to reduce both of these parameters, CDNs traditionally place servers as close to the edge networks that users are on as possible. Theoretically the closer the content the faster the delivery, although network distance may not be the factor that leads to best performance. End users will likely experience less jitter, fewer network peaks and surges, and improved stream quality - especially in remote areas. The increased reliability allows a CDN operator to deliver HD quality content with high Quality of Service, low costs and low network load. CDNs can dynamically distribute assets to strategically placed redundant core, fallback and edge servers. CDNs can have automatic server availability sensing with instant user redirection. A CDN can offer 100% availability, even with large power, network or hardware outages.

CDN technologies give more control of asset delivery and network load. They can optimize capacity per customer, provide views of real time load and statistics, reveal which assets are popular, show active regions and report exact viewing details to the customers. These usage details are an important feature that a CDN provider must provide, since the usage logs are no more available at the content source server after it has been plugged into the CDN, because the connections of end-users are now served by the CDN edges instead of the content source.

III. TECHNOLOGY

CDN nodes are usually deployed in multiple locations, often over multiple backbones. These nodes cooperate with each other to satisfy requests for content by end users, transparently moving content to optimize the delivery process. Optimization can take the form of reducing bandwidth costs, improving end-user performance, or increasing global availability of content. The number of nodes and servers making up CDN varies, depending on the architecture, some reaching thousands of nodes with tens of thousands of servers on many remote PoPs. Others build a global network and have a small number of geographical PoPs. Requests for content are typically algorithmically directed to nodes that are optimal in some way. When optimizing for performance, locations that are best for serving content to the user may be chosen. This may be measured by choosing locations that are the fewest hops, the fewest number of network seconds away from the requesting client, or the highest availability in terms of server performance (both current and historical), so as to optimize delivery across local networks. When optimizing for cost, locations that are least expensive may be chosen instead. In a optimal scenario, these two goals tend to align, as servers that are close to the end user at the edge of the network may have an advantage in performance or cost. The Edge Network is grown outward from the origin/s by further acquiring (via purchase, peering, or exchange) co-locations facilities, bandwidth and servers.

IV. CONTENT NETWORKING TECHNIQUES

The Internet was designed according to the end-to- end principle. This principle keeps the core network relatively simple and moves the intelligence as much as possible to the network end-points: the hosts and clients. As a result the core network is specialized, simplified, and optimized to only forward data packets. Content Delivery Networks augment the end-to-end transport network by distributing on it a variety of intelligent applications employing techniques designed to optimize content delivery. The resulting tightly integrated overlay uses web caching, server-load balancing, request routing, and content services.[2]. These techniques are briefly described below. Web caches store popular content on servers that have the greatest demand for the content requested. These shared network appliances reduce bandwidth requirements, reduce server load, and improve the client response times for content stored in the cache. Server-load balancing uses one or more techniques including service based (global load balancing) or hardware based layer 4–7 switches, also known as a web switch, content switch, or multilayer switch to share traffic among a number of servers or web caches. Here the switch is assigned a single virtual IP address. Traffic arriving at the switch is then directed to one of the real web servers attached to the switch. This has the advantages of balancing load, increasing total capacity, improving scalability, and providing increased reliability by redistributing the load of a failed web server and providing server health checks. A content cluster or service node can be formed using a layer 4–7 switch to balance load across a number of servers or a number of web caches within the network. Request routing directs client requests to the content source best able to serve the request. This may involve directing a client request to the service node that is closest to the client, or to the one with the most capacity. A variety of algorithms are used to route the request. These include Global Server Load Balancing, DNS-based request routing, dynamic metafile generation, HTML rewriting, and any casting. Proximity-choosing the closest service node-is estimated using a variety of techniques including reactive probing, proactive probing, and connection monitoring. CDNs use a variety of methods of content delivery including, but not limited to, manual asset copying, active web caches, and global hardware load balancers.

A. Content service protocols

Several protocols suites are designed to provide access to a wide variety of content services distributed throughout a content network. The Internet Content Adaptation Protocol (ICAP) was developed in the late 1990s to provide an open standard for connecting application servers. A more recently defined and robust solution is provided by the Open Pluggable Edge Services (OPES) protocol. This architecture defines OPES service applications that can reside on the OPES processor itself or be executed remotely on a Callout Server. Edge Side Includes or ESI is a small markup language for edge level dynamic web content assembly. It is fairly common for websites to have generated content. It could be because of changing content like catalogs or forums, or because of personalization. This creates a problem for caching systems. To overcome this problem a group of companies created ESI.

B. P2P CDNs

Although Peer-to-Peer (P2P) is not traditional CDN technology, it is increasingly used to deliver content to end users. P2P claims low cost and efficient distribution. Even though P2P actually generates more traffic than traditional client-server CDNs (because a peer also uploads data instead of just downloading it) it's welcomed by parties running content delivery/distribution services. The real strength of P2P shows when one has to distribute highly attractive data, like the latest episode of a soap opera or some sort of software patch/update in short period of time. Ironically, the more people who download the (same) data, the more efficient P2P is, thus slashing the cost of the peering fees that a CDN provider has to pay due to inter-peer delivery (in comparison to the same amount of data distributed using traditional techniques). On the other hand, the ―long tail‖ type material does not benefit much from P2P delivery schema, since, to gain advantage over traditional distribution models, a P2P-enabled CDN must force storing (caching) data on peers--something that is usually not desired by users and which is rarely enabled.

Contrary to popular belief P2P is not limited to low-bandwidth audio-video signal distribution. There is no technical boundary, built-in inefficiency, or flaw-by-design in peer-to-peer technology to prevent distribution of full HD audio+video signal at, for example, 8 Mbit/s. It's just environmental factors, like low (upload) bandwidth or inadequate computing power in CE devices, that prevent HD material being publicly available in P2P CDNs. (Low bandwidth problems also apply to traditional CDN, though.)

There are some concerns about lack of Quality of Service control over P2P distribution, but these are being addressed by the P2P-Next consortium. Other concerns include security (e.g. modification of content to include malware) and DRM.

V. IMPLEMENTATION

Random Early Detection, or RED, is an active queue management algorithm for routers suited for congestion avoidance. In contrast to traditional queue management algorithms, which drop packets only when the buffer is full, the RED algorithm drops arriving packets probabilistically. The probability of drop increases as the estimated average queue size grows. RED responds to a time-averaged queue length, not an instantaneous one[6].

The RED router calculates the average queue size, using a low-pass filter with an exponential weighted moving average. The average queue size is compared to two thresholds, a minimum threshold and a maximum threshold. When the average queue size is less than the minimum threshold, no packets are *marked. When the average queue size is greater than the maximum threshold, every arriving packet is *marked. This ensures that the average queue size does not significantly exceed the maximum threshold.

When the average queue size is between the minimum and the maximum threshold, each arriving packet is *marked with probability p a , where p a is a function of the average queue size avg. Each time that a packet is *marked, the probability that a packet is *marked from a particular connection is roughly proportional to that connections share of the bandwidth at the router[7].

The general RED algorithm[6]: procedure REDAlgorithm() is avg ← average queue size while packetsarrive

if (minth <= avg AND avg < maxth) Compute pa with probability pb mark the arriving packet

else if (maxth < avg)

mark the arriving packet

endif

end REDAlgorithm

RED algorithm contains two main sub algorithms (parts):

1. For computing the average queue size that determines the degree of burstiness that will be allowed in the routers queue.

2. For calculating the packet-*marking probability that determines how frequently the router *marks packets, given the current level of congestion. The goal is for the router to *mark packets at fairly even spaced intervals, in order to avoid biases and to avoid global synchronization, and to *mark packets sufficiently frequently to control the average queue size.

Average queue size:

The router implements the low pass filter to calculate average queue size. The implemented low pass filter is an exponential weighted moving average (EWMA). RED router computes the average queue size at packet arrivals, rather than at fixed time intervals, the calculation of the average queue size is modified when a packet arrives at the router to an empty queue. After the packet arrives at the router to an empty queue the router calculates m, the number of packets that might have been transmitted by the router during the time that the line was free. The router calculates the average queue size as if m packets had arrived at the router with a queue size of zero[6].

The calculation is as follows:

avg ← (1 − wq) m avg ;when queue is empty

avg ← (1 − wq)avg + wqq ;when queue is not empty

wq : time constant for low pass filter

m : idle time of the queue/transmission time

Upper bound for wq

Making wq too large does not filter out transient congestion at the router. Assume that the queue is initially empty, with an average queue size of zero, and then the queue increases from 0 to L packets over L packet arrivals. After the Lth packet arrives at the router, the average queue size avgL is

avgL= i=1Liwq(1-wq)L-i

= wq(1-wq)Li=1Li(11-wq)i

= L+1+((1-wq)L+1-1)/wq

Given a minimum threshold minth, and given that the bursts of L packets arriving at the router, then wq should be chosen to satisfy the following equation for avgL < minth:

L+1+((1-wq)L+1-1)/wq < minth

Threshold value setting

Two RED parameters, minth (minimum threshold) and maxth (maximum threshold) are used to decide the *marking probability. minth specifies the average queue size below which no packets will be

*marked, while maxth specifies the average queue size above which all packets will be *marked. As the average queue size varies from minth to maxth, packets will be dropped with a probability that varies linearly from 0 to maxp.

The optimal values for minth and maxth depends on the desired average queue. For a bursty traffic the minth threshold value should be large to allow the link utilization to be maintained at an acceptably high level. The optimal value for maxth depends in part on the maximum average delay that can be allowed by the router.

Maxth >= 2(minth)

Packet *marking probability

The packet-*marking probability pb is calculated as a linear function of the average queue size and varies linearly from 0 to maxp:

pb ← maxp(avg − minth)/(maxth − minth).

The final packet-*marking probability pa increases slowly as the count increases since the last *marked packet.

pa

←pb/(1−count•pb)

The implemented RED algorithm measures the queue in bytes rather than in packets. The method implemented in the RED algorithm to calculate the final packet *marking probability is using Uniform Random Variable. When minth ≤ avg < maxth , a new pseudo-random number R is computed for each arriving packet, where R = Random[0,1] is from the uniform distribution on [0,1].

The arriving packet is *marked if :

R<

pb/(1−count•pb).

Since the average queue size changes over time, the algorithm recomputes R/pb each time when pb is computed. Then after *mark the n-th arriving packet if n ≥ R/pb. Detailed Algorithm[6]:

procedure DetailedREDAlgorithm() is

avg ← 0 count ← -1 while

packetsarrive avg ← average queue size

if (queue is not empty) then avg ← (1

− wq)avg + wq q else

m← idle time of queue/transmission time avg←(1− wq)mavg

if minth ≤ avg < maxth

increment count

calculate probability pa:

pb ← maxp(avg − minth)/(maxth − minth) pa ←pb/(1−count•pb)

with probability pa:

*mark the arriving packet

count ← 0

else if maxth ≤ avg

**mark every arriving packet

count ← 0 else count ← −1 endif

end

end DetailedREDAlgorithm.

VI. CONCLUSION

As we have seen the construction of sensors to detect selfish or malicious nodes in ad hoc networks is a complex task. In this paper we have presented a number of different sensors that can detect different kinds of selfish nodes with a good confidence as shown by our simulation results. If multiple sensors are active in parallel and a selfish node is detected by a number of these sensors, then this is a good indication for excluding the node from the network. One remaining problem with our current simulations is that all the thresholds need to be set manually in order to get good detection results. So in the future we will try to find ways how these values can be set and adjusted automatically during operation. Possible candidates might be some kind of an adjustment algorithm or a self-learning system using neural networks. Furthermore we plan to develop and test additional sensors that will e.g. use topology information from the routing protocol in order to detect selfish nodes.

REFERENCES

[1] X. Ao, Report on DIMACS Workshop on Large-Scale InternetAttacks, http://dimacs.rutgers.edu/Workshops/Attacks/i nternet-attack-9-03.pdf, Sept. 2003.

[2] R. Thomas, ISP Security BOF, NANOG 28, http://www.nanog. org/mtg- 0306/pdf/thomas.pdf, June 2003.

[3] K.A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R.A.Olsson, Detecting Disruptive Routers: A Distributed Network Monitoring Approach,‖ Proc. IEEE Symp. Security and Privacy (S&P ‘98), pp. 115-124, May 1998.

[4] A.T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage, ―Detecting and Isolating Malicious Routers,‖ IEEE Trans. Dependable and Secure Computing, vol. 3, no. 3, pp. 230-244, July-Sept. 2006.

[5] L. Subramanian, V. Roth, I. Stoica, S.Shenker, and R. Katz, ―Listen and Whisper: Security Mechanisms for BGP,‖ Proc. First Symp. Networked Systems Design and Implementation (NSDI ‘04), Mar. 2004.

[6] S. Kent, C. Lynn, J. Mikkelson, and K. Seo, Secure Border Gateway Protocol (Secure- BGP),‖ IEEE J. Selected Areas in Comm., vol. 18, no. 4, pp. 582-592, Apr. 2000.

[7] Y.-C. Hu, A. Perrig, and D.B. Johnson, Ariadne: A Secure On- Demand Routing Protocol for Ad Hoc Networks,‖ Proc. ACM MobiCom ‘02, Sept. 2002.

[8] B.R. Smith and J. Garcia-Luna-Aceves, Securing the Border Gateway Routing Protocol,‖ Proc. IEEE Global Internet, Nov. 1996.

[9] S.Cheung,- An Efficient Message Authentication Scheme for Link State Routing,Proc. 13th Ann. Computer Security Applications Conf. (ACSAC ‘97), pp. 90-98, 1997.