DoS Resilience of Real Time Streaming Protocol

Nihat Altiparmak, Ali Tekeoglu, Ali Şaman Tosun
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX, 78249
Email: {naltipar,tekeoglu,tosun}@cs.utsa.edu

Abstract

Denial of Service (DoS) attacks on a computer system or network cause loss of service to users typically by flooding a victim with many requests or by disrupting the connections between two machines. Although significant amount of work has been done on DoS attacks, DoS attacks on streaming video servers were not investigated in detail. In this paper, we investigate DoS resilience of Real Time Streaming Protocol (RTSP). We show that by using a simple command line tool that opens a large number of RTSP connections, we can launch DoS attacks on the server and the proxy. We discuss in detail how the CPU and the memory resources are affected by the attacks. We observe that, with the DoS attack we launch, clients can also keep the connections alive for a long period and maintain the resources allocated at the server. We propose a lightweight dynamic detection framework for the RTSP based DoS attacks.

1 Introduction

Multimedia streaming is real-time transmission of stored media, where the media content is not downloaded in full, but is played out while parts of the content are being received and decoded. Due to its real-time nature, there are timing constraints and the media must be played out continuously. Real Time Streaming Protocol (RTSP) is the standard protocol for a client to interact with the streaming server in order to request and control the streaming of a media. Common streaming servers include Darwin Streaming Server (DSS) from Apple and Helix DNA Streaming Server (HSS) from Real Networks.

Denial of Service (DoS) is an attack on a computer system or network that causes a loss of service to users typically by flooding a victim with many requests or by disrupting the connections between two machines thereby preventing access to service. A DoS attack can be either a single-source attack, originating at only one host, or multi-source, where multiple hosts coordinate to flood the victim with attack packets. The latter is called a Distributed Denial of Service (DDoS) attack. Common forms of DoS attack are SYN attack, Smurf attack, Buffer Overflow attack, Teardrop attack and internet worms. Detection of DoS attacks can be done by tracing back towards the source of the DoS attack [9] and by identifying the network path traversed by the attack traffic [15]. Several approaches for protection against DoS attacks are proposed [16, 17] including probabilistic approaches [1, 8] that reduce the probability of DoS attacks.

Streaming video servers are more vulnerable against DoS attacks since they allocate a lot of resources up front. Examination of the source code shows that for a single media requested, DSS allocates 5-6 MBytes of shared buffer and 128Kbyte to 1 Mbyte of private buffer. Shared buffer is used for caching purposes and private buffer is unique to each connection. The nature of DoS attacks on streaming servers are different as well. Although attacks such as SYN attack can be launched against streaming servers, attacks unique to streaming servers are at the application level and are launched after the Transmission Control Protocol (TCP) connection has been established. Attacks can use RTSP to force the server out of resources by sending large number of streaming requests through an attack tool instead of a media player. In addition, by sending periodic heartbeat messages, the tool can force the server to maintain memory and other resources already allocated.

Contributions of this work are as follows:

- We show that by using a simple tool, it is possible to open hundreds of RTSP connections from a single client to streaming servers.
- We investigate the effect of large number of connections from a single client on the resources of streaming servers and proxies.
- We show that streaming servers allow multimedia players to send some sort of heartbeat messages to keep connections alive.
- We propose a solution that can detect the attacks while differentiating Network Address Translation (NAT) devices and streaming proxies from malicious users.

978-1-4673-0012-4/11/$26.00 ©2011 IEEE

The rest of the paper is organized as follows. In section 2 we briefly describe the RTSP. We discuss attack framework in section 3 and experimental results in section 4. We describe how RTSP connections can be kept alive in section 5. Detection of attacks are discussed in section 6. Finally, we conclude with section 7.

2 RTSP Protocol

RTSP is an application level protocol to control the delivery of data with real-time properties. RTSP enables controlled, on-demand delivery of real-time data, such as audio and video. Delivered data could be live or stored and can be delivered over TCP, UDP or multicast UDP. RTSP establishes and controls single or several time-synchronized streams of continuous media. However, it does not typically deliver the streaming data itself. It behaves like a network remote control for multimedia servers. It is similar to HTTP protocol in syntax and operation with some major differences such that RTSP is a stateful protocol while HTTP is stateless. Besides, RTSP maintains session IDs to keep track of each client. Readers are directed to [12] for more information about RTSP.

3 Attack Framework

In this section, we discuss the attack framework we used including tools to open multiple RTSP connections from a single client, streaming servers, streaming proxies, testbed, and the tools to monitor resource usage.

3.1 Opening Multiple RTSP Connections

openRTSP [10] is an open source, command line tool that can be used to open, stream, receive, and record media streams that are specified by an RTSP URL (begins with *rtsp://*). The program opens the given URL with proper RTSP requests, retrieves the Session Description for each audio and video sub-sessions, and sets up and plays the sub sessions. In our experiments, we used the following combination of options for openRTSP: *openRTSP -c -r -p 6666 <RTSP_URL>*. In this command, *-c* option plays the media continuously and *-r* option with *-p* option enables us to redirect the stream to a specified port. Since no application is attached to the port 6666, the data sent by the server is ignored by the client.

3.2 Streaming Servers and Proxies Used

In our experiments, we used two open source streaming server products; Apple's Darwin Streaming Server (DSS) [3] and RealNetworks' Helix DNA Streaming Server (HSS) [6] together with their proxy servers; Darwin Streaming Proxy (DSP) [2] and Helix Streaming Proxy (HSP) [11]. We chose DSS and HSS because of their open-source and platform independent features; however, since RTSP is the standard protocol that streaming servers implement and our attack is based on RTSP, other streaming servers are also expected to be vulnerable.

3.3 Testbed

We used a testbed consisting of six machines; two streaming servers, two streaming proxies, and two clients connected using two Gigabit Ethernet switches. The testbed is given in Figure 1. A separate switch is used for each client-proxy-server setup. Servers have Dual Pentium 4 3.2Ghz CPUs, 2GB memory and an Intel Gigabit Ethernet connection running on Ubuntu 7.10. Proxies have a Pentium 4 3GHz CPU, 2GB memory and a Broadcam gigabit Ethernet connection running on Ubuntu 7.10. DELL PowerConnect 5212 is used as a gigabit Ethernet switch.

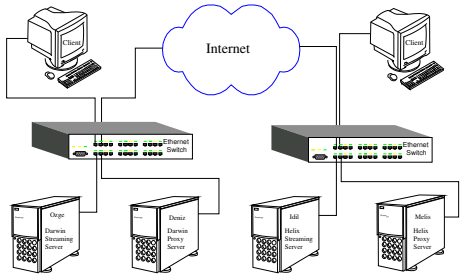


Figure 1. Testbed Used for Experiments

3.4 How to Monitor Resource Usage

We used SYSSTAT [13] tools to monitor CPU and memory use of proxies and servers. In our experiments, we used PIDSTAT tool of SYSSTAT. PIDSTAT is capable of monitoring each individual task in Linux kernel when information about specific task activities are provided using its flags. For our memory experiments, we used *-r* flag of PIDSTAT and collected the data of physical memory used by each specific task. For the CPU experiments, we used *-u* flag to collect the total percentage of CPU used by the task.

4 Experimental Results

We have experimental results using a single client for each experiment. The client opens multiple connections to the streaming proxy or the streaming server depending on whether a streaming proxy is used or not. We focused on memory and CPU usage and investigated how they differ depending on the number of the parameters. We are also interested in how the use of streaming proxy affects the experimental results. To include this we have experimental results for two settings; first setting has a client and a streaming server, second setting has a client, a streaming proxy, and a streaming server.

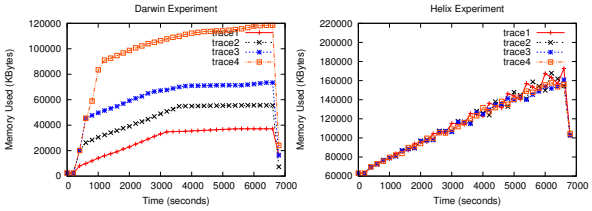
We used 25 movie clips in our experiments and have 4 traces. In each trace, we streamed total of 200 movies continuously. In *trace1*, the same movie is streamed 200 times. In *trace2*, 5 different movies are streamed and this is repeated 40 times. In *trace3*, 10 different movies are streamed and this is repeated 20 times. In *trace4*, 25 different movies are streamed and this is repeated 8 times. The goal in using

different traces is to see how caching affects the results. We varied the delay between opening two successive streaming sessions to see how the delay affects the experimental results. We set the delay to 30, 60, 120 and 300 seconds. We also have experimental results with large number of requests for 0 seconds delay.

We next give a brief summary of experimental results. We put together a set of graphs that are interesting for our purpose. At the end of each experiment, we kill all the RTSP connections opened to release the resources held. This can be observed as a drop at the end of some graphs if OS has enough time to release the resources during the experiment. Note that the results are obtained using only a single client opening 200 connections for investigation purposes; however, opening more number of connections from multiple clients at the same time as in DDoS attack scenario will make the attack severe.

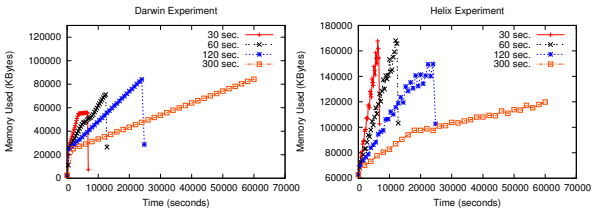
4.1 Streaming without Proxy

- *Memory Use:* When movies are streamed with 30 seconds delay between them, the memory use for all the traces is shown in Figure 2. A single client can force DSS (a) to use up to 120MB of memory and HSS (b) to use up to 170MB of memory. When the movies are streamed with different delays, we see that memory consumption depends on the delay between successive requests. For *trace2* this can be seen in Figure 3. Besides, peak memory use for DSS increases as large number of different movies are streamed. This can be seen in Figure 4(a) for *trace4*. DSS memory consumption jumps to 140MB in this case while HSS memory consumption remains the same in Figure 4(b).



(a) DSS (b) HSS

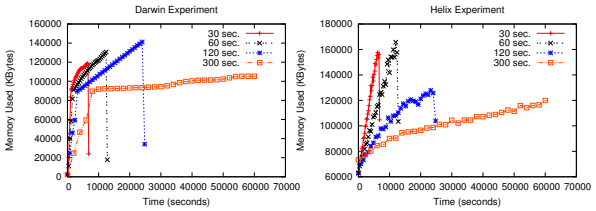
Figure 2. Memory, all traces, 30 sec. delay



(a) DSS (b) HSS

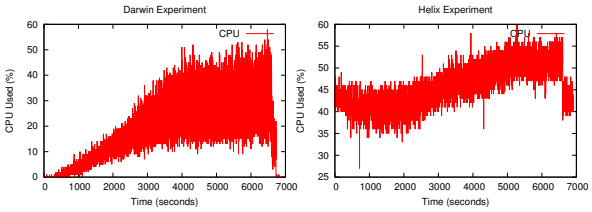
Figure 3. Memory, trace2, all delays

- *CPU Use:* When movies are opened with 30 seconds delay between them, the CPU utilization for *trace4* is shown in Figure 5. A single client can force DSS (a) to use up to 50% of the CPU and can force HSS (b) to use up to 55% of the CPU.



(a) DSS (b) HSS

Figure 4. Memory, trace4, all delays

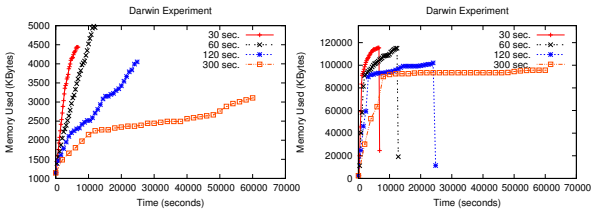


(a) DSS (b) HSS

Figure 5. CPU, trace4, 30 sec. delay

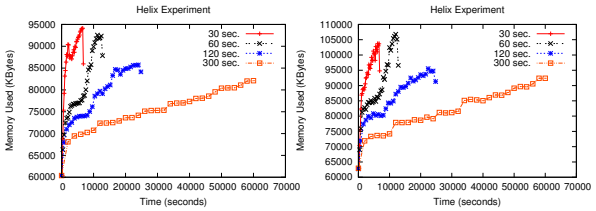
4.2 Streaming with Proxy

- *Memory Use:* For *trace4*, memory consumptions of DSP and DSS are shown in Figure 6(a) and Figure 6(b) respectively. Using the proxy slightly reduces the memory used by the server for some traces but does not have a major effect in general. DSP has minimal memory use since it does not apply any caching. Same data for Helix is shown in Figure 7. HSP (a) reduces the memory consumption of HSS (b) dramatically but HSP uses almost 90% of the memory used by the HSS now.



(a) Proxy (b) Server

Figure 6. Memory, DSP and DSS, trace4

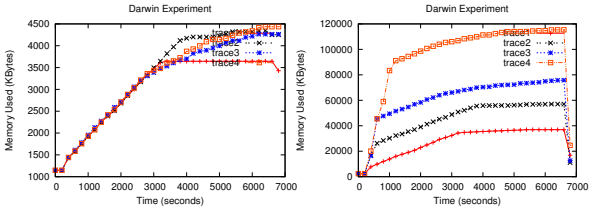


(a) Proxy (b) Server

Figure 7. Memory, HSP and HSS, trace4

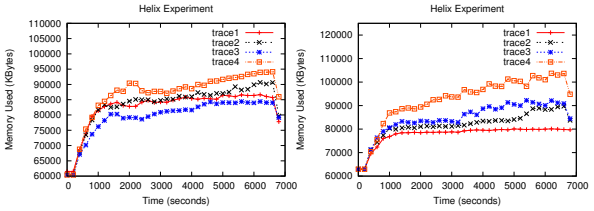
DSP and DSS memory consumption when movies are streamed with 30 seconds delay is shown in Figure 8(a) and Figure 8(b) respectively. Since DSP has

no caching ability, the results for the DSS are similar to the no-proxy case as in Figure 2. DSP uses only 4.5MB of memory on its peak. Same data for Helix is shown in Figure 9. Peak memory use for HSS (b) reduces from 170MB to 100MB with the HSP. However, HSP (a) uses up to 95MB of memory.



(a) Proxy (b) Server

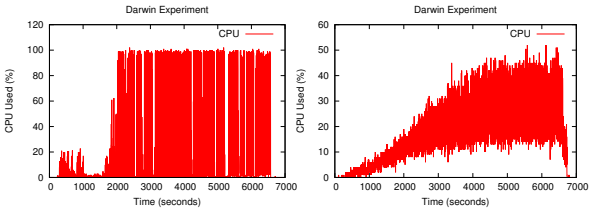
Figure 8. Memory, DSP and DSS, 30 sec.



(a) Proxy (b) Server

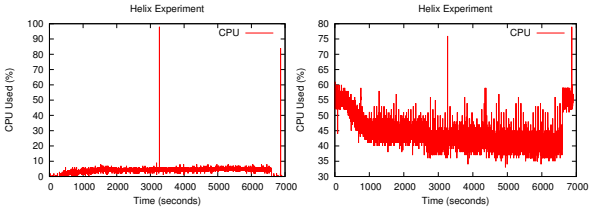
Figure 9. Memory, HSP and HSS, 30 sec.

- *CPU Use:* When movies are streamed with 30 seconds delay between them, the CPU utilization for *trace4* for DSP and DSS is shown in Figure 10(a) and Figure 10(b) respectively. A single client can force DSP to use almost 100% CPU and use up to 50% of the DSS CPU. Same data for Helix is shown in Figure 11. CPU utilization is less than 10% for the HSP (a) and similar to the non-proxy case for the HSS (b).



(a) Proxy (b) Server

Figure 10. CPU, DSP and DSS, trace4

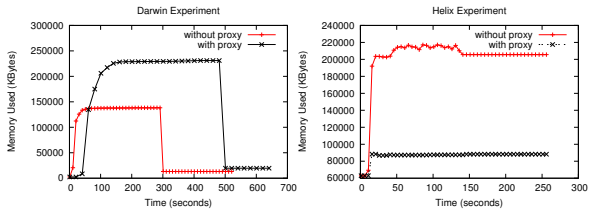


(a) Proxy (b) Server

Figure 11. CPU, HSP and HSS, trace4

4.3 Large Number of Sudden Connections

We investigated how CPU and memory use change when a large number of connections are opened with 0 delay. DSS memory use for *trace4* with and without DSP are shown in Figure 12(a) for 1000 sudden connections. Memory use for DSS is higher when DSP is used. The reason for this is because of the number of connections opened. Without DSP, DSS opened 183 connections. However, DSS was able to open 510 connections with DSP. Without DSP, DSS gave *453 Not Enough Bandwidth* error as a response to the RTSP-Describe request of the client and DSS could not open the connection. This is why the number of connections is lower without DSP.



(a) DSS (b) HSS

Figure 12. 1000 Connections, 0 delay

Same data for Helix is shown in Figure 12(b). Memory use without HSP is higher in this case as expected. When 1000 connections are requested at the same time, HSS was able to open about 900 of them for a short time period. But then number of connected clients are declined to 100. This was because of the bandwidth availability of HSS which is connected to a 1Gbps switch. In other words, HSS was able to accept 900 of 1000 requests but when it starts streaming, due to bandwidth limitation it drops the number of clients to about 100. Besides bandwidth, we observed that HSS could not maintain connections because after a while it run out of file descriptors and gave the error *open() failed with too many open files* for each connection request. When we conduct the same experiment with HSP in front of the HSS, the number of clients dropped to about 300. This was because of HSP's limitation on bandwidth. So, one out of 3 requests is denied by the HSP in this experiment.

4.4 Discussion

In this section, we will share some observations we made out of the experimental results given above. First of all, Figure 2(a) shows that in non-proxy case streaming different movies increases the memory use of Darwin Streaming Server although it does not have much effect on Helix Streaming Server in Figure 2(b). We can also see the incremental effect of streaming different movies on the memory use of Darwin Streaming Server from Figure 3(a) and Figure 4(a). In Figure 3(a), the peak memory use of Darwin Streaming Server is 80MB when it is 140MB in Figure 4(a). However, the results for Helix Streaming Server in Figure 3(b) and Figure 4(b) are again similar without depending on the content of the request. Besides, Helix Streaming Server uses about 40% more memory than Darwin Streaming Server for *trace4* and nearly 4 times more memory for *trace1* in Figure 2. These results clearly show

that Darwin Streaming Server uses some kind of a caching mechanism to reduce its memory use and as a result requires less memory than Helix Streaming Server in general. On the other hand, the situation is exactly opposite for their proxies. We see from Figure 8(b) and Figure 2(a) that using proxy for Darwin Streaming Server does not really effect its memory use, however; Figure 9(b) and Figure 2(b) clearly show that the memory use of Helix Streaming Server decreases nearly 45% in the existence of proxy. In addition to this, the peak memory use of Darwin proxy is about 4.5MB in Figure 8(a) when it is nearly 100MB for Helix proxy in Figure 9(a). As a consequence, we can state that now, Helix proxy uses some kind of a caching mechanism to help its streaming server. The proxy and the server of Helix basically share the total memory usage of the server alone in the non-proxy case. In summary, caching is applied in the server side for Darwin while it is on the proxy side for Helix. In addition to these, although Darwin proxy does not have any caching mechanism, section 4.3 shows that using a streaming proxy in high load enables both of the servers to handle more number of connections by providing a better service to their clients.

5 How to keep connections alive

In addition to opening a large number of RTSP connections from a single client, a client can also keep the connections alive for a long period and maintain the allocated resources at the server. RTSP connections have a default timeout value of one minute. However, server can set a different timeout value in Session Response Header. Server needs ”wellness” information from clients to keep the connection alive within this timeout. If the server does not receive any keep-alive packets such as RTCP reports or RTSP commands (GET_PARAMETER, SET_PARAMETER or OPTIONS) from the client, it terminates the connection and releases the resources reserved for that client.

We have used several media players to establish streaming sessions with DSS and HSS, and analyzed the messages exchanged between client and server using the wireshark [1] tool. HSS declares its timeout value in SETUP response, but DSS does not reveal its timeout value in RTSP responses; so clients have to keep connection alive with RTCP reports. openRTSP sends RTCP report messages to server with about 4 to 6 second intervals to keep the session alive. Similarly, VLC Media Player also uses RTCP report messages instead of sending RTSP methods. Since Real Player is implemented by Real Networks and HSS defines its timeout value in the SETUP response header, Real Player prefers to send OPTIONS method to HSS every half of the timeout seconds if there is no other request sent during that time. QuickTime Player is Apple’s media player and it implements RTCP RR (Receiver Report) messages for keeping the connection alive. DSS sends RTCP SR (Sender Report) messages to client and QuickTime player replies to those messages, consequently connection between client and server does not time out.

[1] http://www.wireshark.org

6 Detection of Attacks

DoS attacks in streaming servers differ from the usual DoS attacks by its nature. Most of the known DoS attacks take place during the TCP connection establishment process such as SYN flood attack, which is an attempt to open so many TCP connections by sending TCP SYN packets to the target system. However, DoS attacks unique to streaming servers take place at the application level. In SYN flood, since the connection does not have to be established fully, IP spoofing is a common approach used by attackers and obviously, IP spoofing makes the detection of the attacker more complicated. In streaming servers, a unique way to attack is opening and maintaining so many RTSP connections in order to make use of server’s resources excessively. For this reason, an attacker first has to establish a TCP, then an RTSP connection with a legitimate IP address to be able to receive the packets sent from the server during the communication process. Therefore, IP spoofing is not an option.

DoS attacks unique to streaming servers can be eliminated by enforcing an authentication policy such that each client has to have a legitimate account before streaming a media. However, enforcing authentication is not practical for streaming servers because of the overhead it brings on the client and on the server side. First of all, signing up requirement to stream a media is inconvenient for the client. Besides the inconvenience, authentication will bring the privacy concerns with it such that most of the clients will be uncomfortable with the idea of being monitored. Secondly, handling the user accounts will cause an extra overhead on the server side by keeping up with users and ensuring their privacy. Therefore, our proposed solution works without the requirement of user authentication.

Our framework uses a *Counting Bloom Filter*, a *NAT/Proxy/ Single User Detector* and a *Delta Queue* structure. *Counting Bloom Filter* is used to maintain the number of connections opened by each IP address. *NAT/Proxy/Single User Detector* is used to differentiate the client IP address whether it belongs to a NAT device, a streaming proxy or a single user. And finally, *Delta Queue* is used to check the expiration time of each connection in order to prevent keep alive attack explained in section 5. As in the attack case, our detection scheme is also based on RTSP and expected to work with any streaming server not depending on any implementation details.

6.1 NAT/Proxy/Single User Detector

One of the challenges in detection is how to handle streaming proxies and Network Address Translation (NAT) devices. If a client uses a streaming proxy to connect to a

streaming server, streaming server cannot see the IP address of the client since proxy replaces client's IP address with its own IP address. In other words, streaming server sees the IP address of the proxy in the case of proxy usage. In order to handle multiple clients from a streaming proxy, a higher connection limit should be set for the proxies. Similarly, NAT devices allow multiple clients to use the same IP address by replacing clients' IP addresses with its own. As in the proxy case, a higher connection limit should be set for NAT devices as well. First of all, we will explain how the detection of NAT devices work and then we will discuss the detection of streaming proxies. If an IP address is not classified as either a NAT device or a streaming proxy, it will be assumed to be a single user.

6.1.1 NAT Device Detection

For this purpose, we have a previous research focused on approximating the number of machines behind a device that does network address translation [14]. We use TCP timestamp option and clustering to identify the approximate number of computers behind a NAT device. Timestamp option includes current timestamp of the machine in the TCP packet, which is incremented in certain periods by different operating systems with respect to time. Our scheme works online by clustering timestamps of received packets into lines using least-squares line fit. It maintains convex hull of timestamp points to determine the quality of the clusters with minimal amount of timestamp points. Therefore, after clustering process each cluster looks like a straight line and corresponds to a computer. We implemented our NAT detector and tested against a synthetic workload. In order to create a synthetic workload, we implemented a packet sending tool which is able to generate outbound network traffic of hundreds of machines by sending crafted packets. In Figure 13, we plotted actual number of machines sending packets (#Machine Sent), number of machines sending at least threshold number of packets (#MachinesOverThreshold) and the number of machines detected by our NAT detection scheme (#Machine Detected). In order for our detection to work, machines should send at least threshold number of packets, which is set to 5 for this experiment. As shown in the Figure 13, NAT device detection scheme closely approximates the number of machines using their TCP timestamp information.

6.1.2 Streaming Proxy Detection

Recently, proxy detection become a critical component for online fraud detection. Several tools [5, 7] have been developed to detect a proxy. Given the IP address, a proxy detector can understand whether the IP address belongs to a proxy or not. Proxy detectors employ several methods; one commonly used method involves looking for HTTP message-header fields that are only used by proxies. For instance, one of these additional header fields is stored inside the HTTP_X_FORWARD element and is used by various proxies to submit the IP of the client using it. However, in some cases proxy might not use these fields to improve their stealth property. In that case, proxy detection tools can examine the ports that packets are coming from. Some common proxy ports are 8080, 80, 6588, 8000, 3128, 553, 554. Streaming proxy listens for RTSP connections at port number 554 just like the streaming server. Moreover, some proxies add proxy identification information to the RTSP header of the packet such as HSP. In our case, we needed to detect streaming proxies in order to allocate resources accordingly. Thus, we used RTSP header and port number to detect streaming proxies.

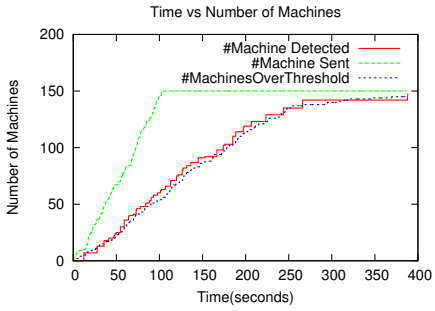


Figure 13. NAT Device Detection Experiment

Although we need to set higher connection limit for proxies, not all the proxies require the same number of connections as a limit. Employing a reputation system allows us to determine these connection limits dynamically. Using a specific reputation algorithm to dynamically compute the reputation scores, system can decide if a proxy is legitimate or not. The reputation scores can be determined by observing the number of connections opened by the proxy for each time interval T. Proxies passing the given limit unexpectedly for consecutive intervals receive a bad score by the reputation system.

6.2 Bloom Filter and Delta Queue

We used a Bloom Filter based counting scheme to maintain the number of connections opened by each IP address. A Bloom Filter (BF) [4] computes k distinct independent uniform hash functions. Each hash function returns an m-bit result and this result is used as an index into a 2^m sized bit array. The array is initially set to zeros and bits are set as data items are inserted. In Counting Bloom Filters (CBF) there is a counter instead of bits and insertion increments the counters corresponding to k hashed values.

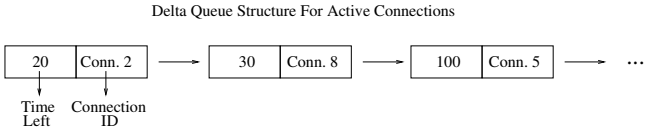


Figure 14. Delta Queue

In order to prevent malicious users maintaining the allocated resources of the streaming server as in Section 5, we used a Detla Queue structure. Delta queue is a type of

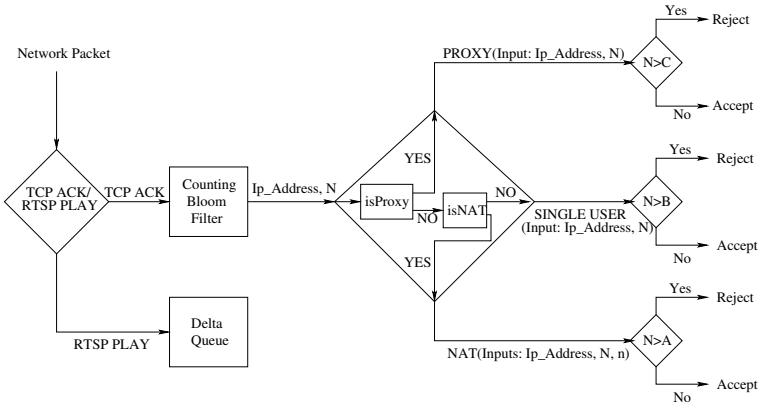


Figure 15. Detection framework

a deadline queue, where each node is basically sorted by their deadlines (expiration time of a connection). In Figure 14, there is a sample deadline queue, in which the first node, *Conn. 2*, has 20 seconds to expire, and the second node *Conn. 8* has $20 + 30 = 50$ seconds to expire etc.

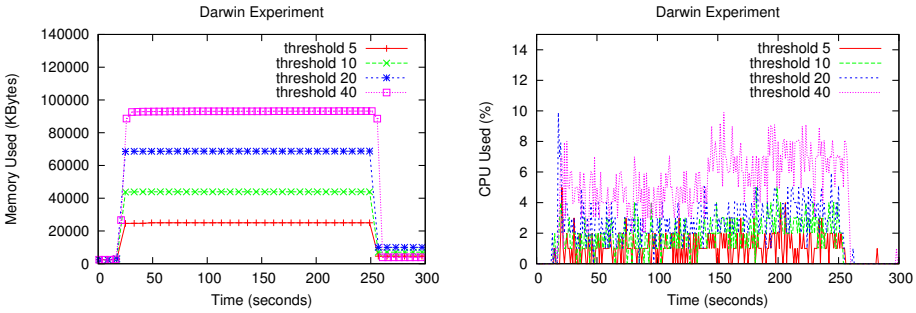
6.3 Proposed Framework

Our proposed framework is given in Figure 15. Each network packet coming to the streaming server is checked whether it is a TCP ACK packet(3^{rd} stage of TCP three-way handshaking) or RTSP PLAY request packet. If the packet is an RTSP PLAY request, then the Delta Queue structure is updated accordingly. For each connection, Delta Queue keeps the time that connection should be alive, $timeAlive$ value. $timeAlive$ is calculated by multiplying the duration of the media requested, $mediaDuration$, and the server load value M such that $timeAlive = mediaDuration \times M$. M is a variable starting from c_1 when the server is busiest and increases until c_2 as the total number of connections for the server decreases. It is reasonable to set c_1 greater than 1 since each connection should be alive until the media requested is played at least once. When a connection is expired in Delta Queue, then the streaming server behaves as if a TEARDOWN request is received from the client and finalizes the session. By this way, we can prevent the malicious users who are trying to hold server's resources by keeping their connections alive as it is explained in section 5. For each session, only the first RTSP PLAY request updates the Delta Queue since a malicious user may try to send fake RTSP PLAY requests to update the Delta Queue and hold the resources of the server. If the packet is TCP ACK, then the Counting Bloom Filter(CBF) is updated accordingly. CBF maintains counters for a period of time T keeping the information of how many RTSP connection is established from each IP address. The goal in having separate counters is to have higher limits for NAT devices and streaming proxies. NAT devices are allowed A number of RTSP connections in period T, and a single user is allowed B number of RTSP connections in period T. Since we know the approximate number of devices behind a certain NAT device, n, the number A changes depending on n for each IP address such that $A = n * B$. Proxies are allowed C number of connections. When period T expires, counters are reset to 0 and a new interval starts. The idea is to have some type of decaying counters and have a dynamic system that enforces restrictions temporarily. Our goal is to have minimal administrator intervention. The values of M, c_1, c_2, T, B and C are environment dependent and can be dynamically configured depending on the load on the system and number of connections. T also depends on the behavior of streaming server. Most commercial streaming servers have a duration limit such that a movie cannot be longer than their pre-set value, which limits the users' streaming duration for a single connection. It is better to configure T by using this pre-set duration limit since incorrect configuration may weaken the detection scheme. Use of CBF and Delta Queue creates a lightweight and space efficient data structure.

6.4 Evaluation

We implemented the proposed detection framework and tested it to see the ability of the framework to control the server's resources. The experiment is performed for a single client suddenly requesting multiple RTSP connections using *trace4* from DSS. The detection software implemented in Java stays in the server side and controls the number of connections opened from a single client by checking each TCP packet coming to the server. The experiment is conducted for one time interval T and different values of B. Memory and CPU use of the streaming server is given in Figure 16(a) and Figure 16(b) respectively for B equals to 5, 10, 20 and 40. As it is clear from the figure, controlling the number of connections from a single client limits the resource use of the server. Higher thresholds let more connections go through and lead to more resource usage. The detection scheme performs similar for HSS; however, the results were omitted due to space restrictions.

The overhead imposed by our detection scheme arises from the three parts of the detection; counting bloom filter, delta queue and NAT detection. Regardless of the size of



(a) Memory (b) CPU

Figure 16. Memory and CPU use with the detection framework

the elements, bloom filter only uses a constant number of bits, c, for each element. Therefore, the memory used by the bloom filter is constant not depending on the number of connections in the system. Running times of the insert and look up operations of bloom filters also require O(c) operations without depending on the number of connections in the system. For the delta queue, we need 6 bytes for each connection; 1 byte to keep the expiration time of the connection, 1 byte for the connection ID and 4 bytes for the pointer. For n number of connections, we need $6n$ bytes space. Insertion operation of delta queue costs O(n) time and deletion costs O(1) time. NAT detection needs to allocate 1 KByte of memory space for buffer and additional 200 Bytes for each detected machine's convex hull. In case there are a thousand machines in an exemplary setup, total memory usage would be only 210 KBytes, which is reasonable. CPU usage is significant only at the time of clustering when the buffer gets full, which requires O(n) time operation for n machines. Readers are directed to [14] for more information on the NAT detection scheme and its overhead.

7 Conclusion

We show that it is possible to open a large number of RTSP connections from a single client to Darwin and Helix streaming servers. Large amount of resources can be blocked at the streaming servers with these connections. Using a testbed we analyzed how memory and CPU resources of the streaming servers and proxies are affected by the large number of RTSP connections. We propose a dynamic and lightweight detection framework using bloom filters. Proposed framework allows temporary restrictions to clients and special handling for NAT devices and proxies. Future work includes integration of proposed detection framework into Darwin proxy and server and Helix proxy and server.

8 Acknowledgments

This research was partially supported by ARO Grant W911NF-11-1-0170 and NSF grant CNS-0855247.

References

[1] T. Anderson and T. Roscoe. Preventing internet denial-of-service with capabilities. In *ACM SIGCOMM Computer Communication Review*, volume 34, 2004.

[2] Apple, http://docs.info.apple.com/article.html?artnum=106306. *Darwin Proxy Server*.

[3] Apple, http://developer.apple.com/opensource/server/streaming. *Darwin Streaming Server*.

[4] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[5] DetectAProxy, http://www.detectaproxy.com/. *Stop Anonymous Proxy Users*.

[6] Helix Community, https://helix-server.helixcommunity.org/. *Helix DNA Streaming Server*.

[7] IP2Proxy, http://www.ip2proxy.com/. *The IP2Proxy Proxy Detection Web Service*.

[8] A. D. Keromytis, V. Misra, and D. Rubenstein. Sos: secure overlay services. In *ACM SIGCOMM Computer Communication Review*, volume 32, August 2002.

[9] J. Li, J. Mirkovic, M. Wang, P. Reiher, and L. Zhang. Save: Source address validity enforcement protocol. In *Proceedings of INFOCOM*, volume 3, pages 1557–1566, 2002.

[10] Live Networks, http://www.live555.com/openRTSP/. *openRTSP*.

[11] Real Networks, http://licensekey.realnetworks.com/rnforms/products /servers/eval/index.html?ulf=g. *Helix Proxy Server*.

[12] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326, Apr. 1998.

[13] SYSSTAT, http://pagesperso-orange.fr/sebastien.godard/index.html. *Collection of System Monitoring Tools for Linux*.

[14] A. Tekeoglu, N. Altiparmak, and A. S. Tosun. Approximating the number of active nodes behind a nat device. In *20th IEEE International Conference on Computer Communications and Networks (ICCCN 2011)*, Maui, Hawaii, August 2011.

[15] A. Yaar, A. Perrig, and D. Song. Pi: A path identification mechansim to defend against ddos attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2003.

[16] A. Yaar, A. Perrig, and D. Song. A stateless internet flow filter to mitigate ddos flooding attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2004.

[17] D. K. Yau, J. C. Lui, F. Liang, and Y. Yam. Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles. In *IEEE/ACM Transactions on Networking*, volume 13, Feb. 2005.