Defending Network-Based Services Against Denial of Service Attacks

Jinu Kurian
Dept. of Computer Science
University of Texas at Dallas
Email: jinuk@student.utdallas.edu

Kamil Sarac
Dept. of Computer Science
University of Texas at Dallas
Email: ksarac@utdallas.edu

Kevin Almeroth
Dept. of Computer Science
University of California, Santa Barbara
Email: almeroth@cs.ucsb.edu

Abstract— **Over the last decade, several value-added services have been proposed for deployment in the Internet. IP multicast is an example of such a service. IP multicast is a *stateful* service in that it requires routers to maintain state for forwarding multicast data toward receivers. This characteristic makes the service and its users vulnerable to denial-of-service (DoS) attacks. One type of attack aims to saturate the available buffer space for storing state information at the routers. A successful attack can prevent end systems from properly joining multicast groups. In this paper, we present a solution to state overload attacks; evaluate the overhead of the solution through a combination of simulation and implementation; and outline an incremental deployment strategy for its partial deployment. The evaluation results indicate that our solution improves the resistance of IP multicast to state overload attacks.**

I. Introduction

Over the last decade, several *value-added* services have been proposed for deployment in the Internet. These include multicast communication [1], quality-of-service support [2], content distribution networks [3], and denial-of-service (DoS) defense mechanisms [4]. These services provide users with an array of added capabilities. They also provide ISPs with an opportunity to provide a new set of services to draw additional revenue. Compared to the stateless nature of the traditional best effort IP packet forwarding service, some of the above mentioned value-added services introduce additional overhead into the network. When misused, this overhead can be a means to launch DoS attacks on the service or its users. In this paper, we take IP multicast as an example and demonstrate how it can be misused to create DoS attacks on the service and its users. We then propose a solution to defend the IP multicast service from these attacks. One high level lesson that we take from this study is the realization of the difficulties in introducing value-added network services without creating a significant level of additional overhead and security vulnerability for the network and its users.

IP multicast is one of the first value-added services to be developed and partially deployed in the Internet [1]. Despite the well known advantages of IP multicast in supporting multi-receiver network applications, the existing multicast protocols suffer from various security flaws that have restricted the use of IP multicast on a larger scale [5], [6]. One important security threat in IP multicast is the possibility of DoS attacks against multicast-enabled routers. DoS attacks are possible because of the additional overhead required for packet forwarding.

The current protocol to build and maintain multicast trees is Protocol Independent Multicast (PIM) [7]. In PIM, in response to join requests coming from multicast receivers, routers create and maintain state entries in exhaustible forwarding state buffers. This mechanism makes routers vulnerable to DoS attacks called *state overload* attacks [6].

State overload attacks can be classified by the intended victim of the attack, either end system or the infrastructure itself. In a *directed end system attack*, the objective is to thwart an end system or its subnet from sourcing or receiving multicast content. By overloading the state buffers at routers in its vicinity, a DoS attack can be executed against a multicast source (e.g. an Internet TV station) preventing new customers from joining and receiving data. In an *infrastructure attack*, the attack target may be a group of the core routers in the network backbone. If successful, the infrastructure attack may have an impact on a large number of multicast users competing for the limited state buffers at the attack point. Both types of attacks are relatively easy to launch and can significantly impact the availability of multicast for end users.

One basic idea to defend against state overload attacks is to rate limit the number of join requests originating from end hosts or multicast enabled subnets [6] [8]. Rate limiting can be effective against state overload attacks that involve one or more attack hosts within the same subnet. However, rate limiting without knowledge about which join requests are valid can have an adverse effect on legitimate join requests. Furthermore, it may not be effective if the attack is sufficiently distributed. Defending against state overload attacks has been partially addressed in a few previously proposed solutions. MAFIA uses distributed group membership information to realize multicast access control and traffic filtering [9]. The Multicast Control Protocol (MCOP) aims to control user access to multicast services in the intra-domain [10]. Both approaches attempt to control the multicast usage (i.e., sourcing or receiving multicast content) of local end users.

In this paper, we propose a proactive solution to defend against state overload attacks. The objective of our solution is to protect multicast-enabled routers from being overloaded with unwanted state information. We introduce certain enhancements to the PIM join procedure to enable routers to verify the validity of a join message before creating state. We evaluate the added overhead and the effectiveness of our approach using a combination of simulation and implementation. Based on our evaluations we observe that our solution is highly effective in preventing state overload attacks while introducing only minimal overhead in the network.

The rest of this paper is organized as follows. Section II summarizes how the PIM-Join mechanism works and why it is vulnerable to attack. Section III describes our modified protocol including its operation, its evaluation, and partial deployment strategies. Finally, Section IV concludes the paper.

II. Problem Description

In this section, we briefly examine how PIM joins work and why the existing procedure is vulnerable to DoS attacks.

A. PIM Join Mechanism

PIM supports two types of join operations: (1) shared tree joins, and (2) source specific joins. Shared tree joins are used to establish a shared tree between the receivers and a pre-selected special router, called the Rendezvous Point (RP). Since the RP is a domain-local router, the join message and the state created are also local. Hence, state overload attacks using shared tree joins can have only a limited, localized effect. In source specific joins, the receivers join directly to the multicast source, S, of a group, (S,G). Since S can be located anywhere in the Internet, attacks via source specific joins are extremely potent, impacting potentially any local or remote victim sites. Therefore we focus on source specific join attacks in this paper.

In PIM, the designated router of a receiver, DR(R), creates a new Join(S,G) message and forwards it towards the designated router of the source, DR(S). All the routers between DR(R) and DR(S) create forwarding state for the (S,G) group as the join message propagates towards S. Routers forward join messages on their shortest path interface towards the source. This interface is called the *incoming interface* (*iif*) or *reverse path forwarding* interface (Int_{RPF}) for the group. For each *iif* entry, the router also includes all the interfaces from which it received PIM-Join messages in an *outgoing interface list* (*oif*) for the group. A router needs to create new forwarding state for a PIM-Join for each distinct (S,G) group. In source specific multicast, a single source can support up to 2^{24} multicast groups, all of which can be used to generate distinct Join(S,G_i) messages.

The forwarding state created is "soft", i.e. it expires if no refresh message arrives from downstream. Each state entry is associated with an *entry-timer* (ET) and each interface in the *oif* for the entry is associated with an *oif-timer* (OT). When the router receives a PIM-Prune message on an interface, *i*, in *oif*, it removes *i* from *oif*. Alternatively, if no refresh message arrives on i during a Join Hold Time period (the default is 260 seconds), OT_i expires and i is removed from *oif*. Finally, when *oif* becomes empty and the ET expires, the router sends a Prune(S,G) message on its *iif* interface for the group and leaves the multicast tree.

B. State Overload Attacks

A vulnerability in the existing PIM-Join procedure that is exploited in a state overload attack is that DR(R) issues a PIM-Join message without verifying whether the source or the group exist. The Join(S,G) message (legitimate or bogus) propagates towards S creating forwarding state at all routers on the R-to-S path. Since the routers do not have any mechanism to verify the validity/existence of the source or the group, they will maintain the (S,G) state as long as R, who could be an attacker, sends a refresh message.

Consider an attack scenario that proceeds in rounds. For the first round of 260 seconds (or the local default Join Hold Time), the attacker generates distinct bogus join messages to create unwanted state information in the routers. In subsequent rounds, it sends refresh messages to continue to maintain the state at the routers. If there are multiple attackers, the amount of state information maintained at routers could be prohibitively large. For example, in a directed end system attack, if there are 5,000 zombies (attackers), with each zombie issuing 10 separate join requests per minute, at the end of the first round, the routers at the target site will need to store more than 200,000 different multicast entries. Similarly, an infrastructure attack can be launched with attackers choosing highly used paths to target routers at the core of the network. In this case, the attacks are less targeted and the state created would potentially be distributed among multiple core routers. Considering the same attack parameters as before, in the worst case, a core router may end up storing up to 200,000 different entries. In both attacks, the number of entries that can be created is prohibitively large. The above discussion suggests that a solution to the problem must involve a verification of the validity of the source and the groups being subscribed to in a join message. We expect that the routers are suitably provisioned to handle the legitimate join requests so that joins to legitimate groups will not create an attack. In the rest of this paper, we present an approach to include such a verification mechanism in the multicast join process.

III. Modified Multicast Join Procedure

A. Modified PIM Joins

The objective of the modified join procedure is to ensure that before creating any state, every router in the forwarding path can individually verify the validity of the source and the group being subscribed to in the join. This verification ensures that bogus join messages sent by malicious receivers cannot create unwanted state in the routers.

During join forwarding, routers do not create any forwarding state, but instead add the requisite state information to the join message before sending it upstream towards the source. Each on-tree router, R_j, appends this state information as a nonce, say N_j, to the end of a nonce block in a new Join(S,G,N) message. The state information added includes the incoming interface, i_j, of the join message and a secure hash of all the locally added state, H_j (a detailed description of the state added is deferred to Section III-B).

If the source and the group in the Join(S,G,N) message are valid, the accumulated state information is returned by DR(S) in a new JoinACK(S,G,N) message. Each router, R_j, in the return path *individually* verifies the JoinACK(S,G,N) by recomputing the secure hash H_j with the relevant state information in

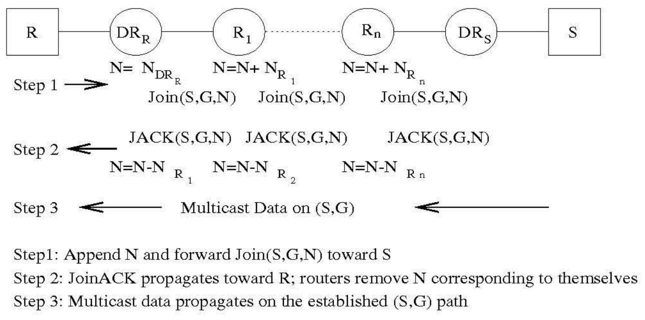

R
DR_R
R_1
R_n
DR_S
S
Step 1
N= N_DR_R
N=N+ N_R_1
N=N+ N_R_n
Join(S,G,N)
Join(S,G,N)
Join(S,G,N)
Step 2
JACK(S,G,N)
JACK(S,G,N)
JACK(S,G,N)
N=N-N_R_1
N=N-N_R_2
N=N-N_R_n
Step 3
Multicast Data on (S,G)
Step1: Append N and forward Join(S,G,N) toward S
Step 2: JoinACK propagates toward R; routers remove N corresponding to themselves
Step 3: Multicast data propagates on the established (S,G) path

Fig. 1. Modified PIM-Join procedure.

the nonce N$_j$. This ensures that the received JoinACK(S,G,N) is a valid acknowledgment of the Join(S,G,N) that R$_j$ had previously forwarded upstream. Once the verification is complete, R$_j$ creates a forwarding entry for (S,G) with i$_j$ as the *oif* and Int$_{RPF}$(S) as the *iif* for the group. Once the JoinACK reaches and is verified by DR(R), the join process is complete.

This operation is visually presented in Figure 1. Figure 2 presents a detailed description of router operation after the proposed modifications to the PIM-Join mechanism. As can be seen from Figure 2, the proposed modifications do not introduce any backward compatibility problems as the routers can process both the existing Join(S,G) and the proposed Join(S,G,N) messages.

The state diagram of a router operating with our modified PIM protocol is shown in Figure 3. It has two states: No-Info (NI) and Joined (J). In the NI state, the router has no knowledge of the existence of a group, i.e., it maintains no (S,G) state about the group. In the J state the router maintains a forwarding entry for (S,G), and keeps an *entry-timer* (ET) and an *oif-timer* (OT) for each interface in *oif* for (S,G). We briefly discuss the state transitions for the routers below:

NI State

In this state, there are two events that cause a router, R$_j$, to take different actions:

- *(1)* **Receiving JoinACK(S,G,N) on interface, k, for a Join(S,G,N):** Verify k = Int$_{RPF}$(S); verify N$_j$ and update N = N - N$_j$; forward JoinACK(S,G,N) if required; create (S,G) state; initialize ET and OT timers; and move to J state.
- *(2-a)* **R$_j$ = DR(S) and receives Join(S,G)/Join(S,G,N):** Verify the validity of (S,G) (by checking with S); create (S,G) state; initialize ET and OT timers; in the case of Join(S,G,N), send a JoinACK(S,G,N) on the interface in *oif*; and move to J state.
- *(2-b)* **R$_j$ $\neq$ DR(S) and receives Join(S,G)/Join(S,G,N):** Add relevant state, N$_j$, to the incoming join message and forward a Join(S,G,N) message toward S on Int$_{RPF}$(S).

J State

In this state there are four events that cause the router, R$_j$, to take different actions:

- **Receiving Join(S,G)/Join(S,G,N) on interface, i:** Refresh its ET for (S,G); set *oif* = *oif* $\cup\{i\}$ and start OT for i. In addition, on Join(S,G,N), send a JoinACK(S,G,N) on i.

```
/* At a router $R_j$ */
    On receiving Join(S,G) on an interface $i$
    IF (S,G) state exists at and ($i \notin$ oif) THEN
        oif = oif = $\cup\{i\}$
    IF (S,G) state NOT exists THEN
        IF $R_j$ = DR(S) THEN
            IF (S,G) group is valid
                Create (S,G) state with iif = Int$_{RPF}$(S); oif = $i$
        ELSE
            Compute N$_j$ and Append N$_j$ to N
            Send Join(S,G,N) on interface $k$ = Int$_{RPF}$(S) toward S

    On receiving Join(S,G,N) on an interface $i$
    IF (S,G) state exists at and ($i \notin$ oif) THEN
        oif = oif $\cup\{i\}$ AND Send JoinACK(S,G,N) on $i$
    IF (S,G) state NOT exist THEN
        IF $R_j$ = DR(S) THEN
            IF (S,G) group is valid
                Create (S,G) state with iif = Int$_{RPF}$(S); oif = $i$
                Send JoinACK(S,G,N) on $i$
        ELSE
            Append N$_j$ to N
            Send Join(S,G,N) on interface $k$ = Int$_{RPF}$(S)

    On receiving JoinACK(S,G,N) on an interface $k$ for a Join(S,G,N) on $i$
    IF k = Int$_{RPF}$(S) THEN
        Recompute and verify N$_j$
        IF (S,G) state NOT exists
            Create (S,G) state with oif = $i$; iif = Int$_{RPF}$(S)
            Modify N = N - N$_j$
            Send JoinACK(S,G,N) on $i$

    On receiving Prune(S,G) on an interface $i$ or OT for $i$ expires
    Remove oif = oif - $\{i\}$
    IF oif is empty THEN
        Send Prune(S,G) on Int$_{RPF}$(S)
        Remove (S,G) state from forwarding state table
```

Fig. 2. Router operation for modified PIM joins.

- **Receiving JoinACK(S,G,N) on interface, k, for a Join(S,G,N) on i:** Verify k = Int$_{RPF}$(S); verify N$_j$ and update N = N - N$_j$; forward JoinACK(S,G,N) on interface i toward R$_{j-1}$. Set *oif* = *oif* $\cup\{i\}$ and start OT for i.
- **Int$_{RPF}$(S) change or join timer expiry:** Send Join(S,G) on Int$_{RPF}$(S). Here R$_j$ issues a Join(S,G) rather than Join(S,G,N) as the (S,G) is already verified prior to the creation of (S,G) state at R$_j$. In this case, Int$_{RPF}$(S) could change due to a unicast routing change. A join timer is used to trigger the transmission of periodic refresh messages upstream.
- **Receiving Prune(S,G) on interface, i, or OT for i expires:** Set *oif* = *oif* - $\{i\}$. If *oif* is empty, send Prune(S,G) on Int$_{RPF}$(S); remove (S,G) state; and move to NI state.

B. Authenticating Joins

In the modified join procedure, every router, R$_j$, adds a nonce, N$_j$, to a Join(S,G,N) message. When the JoinACK(S,G,N) is returned, R$_j$ retrieves N$_j$ for verification and then creates the forwarding state for (S,G). N$_j$ has to carry the requisite information which will allow R$_j$ to create the forwarding state for (S,G). It should also carry path

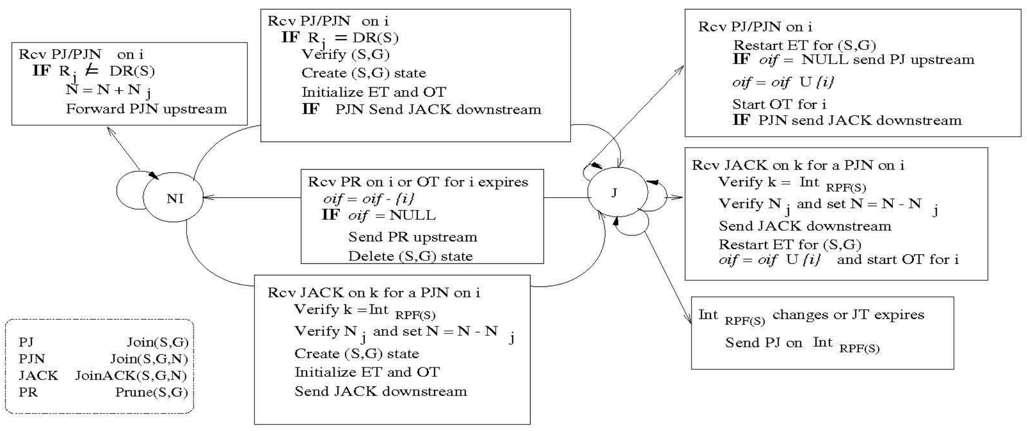


Fig. 3. State machine of a router, R_j, with our modified PIM protocol.

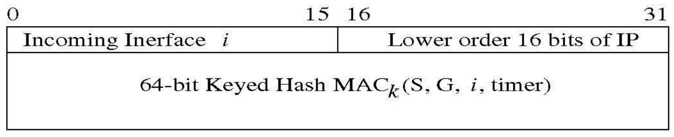

0
15
16
31
Incoming Inerface i
Lower order 16 bits of IP
64-bit Keyed Hash MAC_k(S, G, i, timer)

Fig. 4. Nonce, N_j, added at router, R_j.

information to ensure that the JoinACK(S,G,N) is returned downstream along the same path as the original join upstream. Additionally, the nonce has to be secure against modification, brute force, and replay attacks during its valid duration.

Routers can create this nonce by including (1) the 16-bit incoming interface, i, for the incoming join, (2) the lower order 16 bits of the IP address of the downstream router forwarding the join, and (3) a keyed-hash or MAC of the group address, the incoming join interface, i, and an ascending counter T (see Figure 4). The nonce created is bound to a specific group, (S,G), and interface, i. The nonce, N_j, is then appended to the end of the nonce block, N, in the join message before it is forwarded upstream. The combination of 16-bit interface ID, i, and the lower order 16 bits of the IP address of the downstream router enable the current router, R_j, to derive the reverse path for the JoinACK when it is received later from an upstream router. Note here that the JoinACK will always take the reverse of the original Join path, irrespective of the network level path. A keyed hash like HMAC-SHA or HMAC-MD5 can be used to create a 64-bit hash string in the nonce. The hash creation can be done without modification and without noticeable overhead in most routers. The secret key, k, is randomly generated by the router and can be varied at a rate slower than the clock to provide added security against brute force attacks on the nonce.

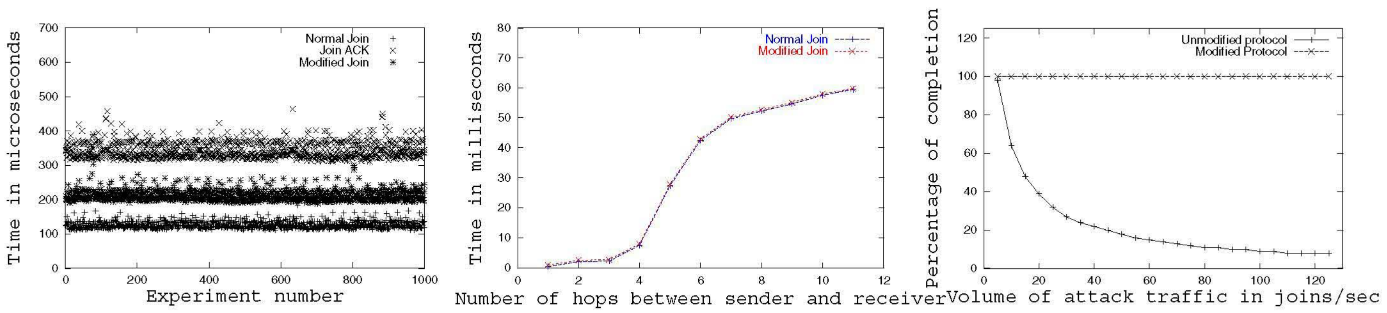
On the return path, when a router, R_j, receives a JoinACK(S,G,N), it first performs an RPF check on the incoming interface of the JoinACK(S,G,N). The RPF check amounts to a lightweight authentication of the upstream router. The RPF check along with the presence of a timer in the hash prevents replay attacks using the same nonce. R_j then extracts N_j from the head of N. N_j includes i and (S,G). The router uses the extracted information and its current (or last few) keys to create a 64-bit keyed hash of (S,G), i, and the current (or last few) counter values. It then authenticates the JoinACK by comparing this hash with the one in N_j that arrived with the JoinACK. After this verification is complete, the router proceeds with the rest of the join process as described in Section III-A.

C. Discussion

We now briefly discuss some important issues with our proposed solution. The modified PIM protocol effectively prevents a receiver from overloading routers with state for bogus (S,G) groups. This feature, however, assumes that the sender is not malicious and is not sourcing bogus groups. In this case, malicious senders and receivers can co-operate with each other to launch an infrastructure attack. To protect against malicious sources, the modified protocol can be combined with source authentication mechanisms [9] at the source's domain. This additional component can ensure the validity of a source and group being subscribed to and prevent these attacks.

Another issue is related to packet fragmentation. Since, in the proposed solution, routers append information to the forwarded join messages, fragmentation possibilities should be considered carefully. The standard PIM-Join message, which includes a single group subscription, is 24 bytes. In our approach, each on-tree router appends 12 bytes to the end of the join message. Therefore, the size of a join message is `24 +(12*n)` where `n` is the number of routers on the path. Assuming `n=40` which is a safe estimate for the Internet today, the maximum size of a modified join message is 504 bytes. Given the fact that IP requires a minimum MTU of 576 bytes, the proposed approach should not cause any fragmentation.

Another issue is the possible loss of JoinACK packets in the network. In such a situation, the routers from the point of loss in the reverse path downstream will not create state entries while state entries will be created upstream. In this scenario, if the receiver does not issue a new Join request, the created states will be dissolved upon timeout. If the receiver issues a new Join request before the timeout, the JoinACK will



(a) Processing overhead at a router. (b) Latency perceived by receiver. (c) Percentage of completed joins.

Fig. 5. Evaluation Results.

be returned by the first router with an established state entry upstream and the Join request will be completed as normal.

D. Evaluation

In this section, we evaluate the overhead introduced by our modified PIM protocol and its performance under DoS attacks.

Processing overhead at a router: We use a Linux-based router to measure the processing overhead in computing and verifying the nonce in the modified joins. For this measurement, we use the sample implementation of HMAC-MD5 from RFC 2104 [11]. For the unmodified version of the PIM protocol, we use the implementation available in the Linux kernel.

Our metric for comparison is the total time taken from reception of a join message to processing and forwarding it upstream. For the modified protocol, we also measured the time taken from reception of a JoinACK from upstream to verify it and send it downstream. The Join requests are generated at a rate of 5000 requests/second and the averaged time taken over 5 seconds was measured. Each experiment was repeated 1000 times and the results are shown in Figure 5(a).

Figure 5(a) shows that per-node processing overhead is about 4 times higher for a modified PIM-Join as compared to a normal PIM-Join. However, from an end-user's point of view, the perceived latency is a more important metric as it indicates the overall performance impact of the modified protocol. The perceived latency includes additional components like queuing and propagation delay at routers. To evaluate the end-to-end latency as perceived by a user, we performed Network Simulator 2 (ns-2) simulations to compare the delay incurred in the modified and unmodified cases. The metric of interest is the total time from the user issuing a PIM-Join request until it starts receiving multicast data from the group. Here, we assume that as soon as the PIM-Join request (modified or unmodified) arrives at the source site, the source starts sending multicast data.

In our simulations, we used the 90th percentile values from the previous experiments as the nodal processing time incurred at on-tree PIM routers. Figure 5(b) presents the results of our simulations. As can be seen from the figure, the total latency, as perceived by an end-user, is virtually identical for both cases. This result is because, in a network, the inter-nodal latency, which is on the order of milliseconds, becomes much more significant than the per-node processing overhead, which is on the order of microseconds. As a result, the end-user perceives very little difference in the delay introduced by these two protocols.

Percentage of completed joins under attack: To evaluate the resistance of the modified protocol to state overload attacks, we performed various experiments in ns-2 using a simulated network topology (see Figure 6). In our experiments, users attempt to join a remote group while the routers on the path are subject to state overload attacks of varying magnitudes. The evaluations are performed under the assumption that there is no loss in the network because the objective of the attack is not to congest the network but to overload the routers. In addition, we assume that the designated router at the source site can distinguish between legitimate and malicious joins based on the group address in the join message. As we showed in Section II, with a distributed attack generated by 5000 zombies, routers in the vicinity of the victim may need to store as many as 200,000 entries. To simplify our simulation, we consider attacks of smaller magnitudes (25 zombies at its peak) and a smaller state buffer threshold (i.e., the number of entries a router can accommodate before it starts dropping new requests).

In the simulations, a legitimate user issues join requests at the rate of 5 joins/sec while the attack traffic is varied from 0 to 125 joins/sec. The results displayed are for a state buffer threshold value of 200 entries. The metric used is the percentage of completed legitimate joins. Figure 5(c) shows the results for the modified and the unmodified protocol. As can be seen in the unmodified protocol, the percentage of completed joins decreases exponentially as the rate of attack increases because the legitimate and the attack traffic compete for the same limited buffer space. The modified protocol meanwhile maintains a 100 % completion rate because only legitimate joins create state. Our evaluation demonstrates that our modified PIM protocol is highly effective in preventing state overload attacks. In addition, the processing overhead required in routers is higher, but it does not cause a noticeable performance degradation for the end user.

E. Partial Deployment Scenario

Our proposed solution requires all PIM routers to be updated to support the modified join operation. In this section, we consider a method which can provide a temporary solution to ISPs supporting our protocol when the neighboring domains do not support the modifications. In this discussion, we refer to routers with the updated PIM protocol as modified routers and the routers employing the non-updated PIM version as legacy routers. For our protocol to function properly, the modified routers require a valid JoinACK message from their upstream neighbors. Downstream routers can be legacy routers without affecting the protocol which will function normally in the domains with modified routers. If a modified router is a domain edge router having a PIM neighborhood relationship with a legacy router of a neighboring domain, it will not be able to receive JoinACK messages.

To deal with such cases, we introduce a proxy-based approach for an ISP supporting our proposed protocol. In this approach, the ISP can deploy *state boxes* at the edges of its domain. The state boxes are high capacity storage devices capable of handling large amounts of data. When an edge router, R_e, of the domain detects (as a result of periodic PIM Hello message exchange) that its next hop neighbor in the neighboring domain is a legacy router, it removes and forwards the accumulated nonce information from the join messages to the local state box. This state is maintained for a small duration of time (e.g., 260 seconds), and is indexed under the appropriate (S,G) value of the incoming join message. The edge router, R_e, then forwards an unmodified join message upstream towards the source.

If the source is valid and is transmitting regularly, data coming from this source will flow down the established path to the edge router, R_e. R_e verifies with the state box if an entry for this (S,G) exists in it. If an entry exists, R_e retrieves the state information from the local state box and issues a JoinACK with the stored state information downstream, thereby establishing forwarding state in the routers in its domain. This state caching operation is visually presented in Figure 7. Until the (S,G) group is verified as valid and the JoinACK is issued, all multicast data for the group from upstream will be redirected by R_e for temporary storage at the state box. After the JoinACK is issued, the buffered multicast data for the (S,G) group is retrieved from the state box and sent downstream along the newly established multicast path towards the receiver. If the source is invalid or has not transmitted for a long period of time, the state is dissolved at the state box to reclaim the state buffer occupied by this state.

In this solution, the state-boxes are a possible point of attack if the attack volume is excessively high. Considering the numbers used previously in Section II and with a worst-case scenario of 40 hops between the attackers and R_e, this could amount to `2.6M*504=48MB` of state information at the end of 260 seconds. Common storage devices are available today with capacities in the range of 40 to 300 GB, so overflowing a state-box with excess state is implausible.

IV. Conclusion

DoS attacks pose a serious problem to the health and security of value-added services in the Internet. In this paper, we have examined DoS attacks, called state overload attacks, for a specific service, multicast. Since the attacks exploit an inherent weakness in the PIM protocol, we proposed a set of modifications to make it more secure against these attacks. The modifications provide an effective solution against DoS attacks without creating noticeable performance loss or latency for the end user. Also, our solution can be incrementally deployed in the inter-domain and can provide an equally effective defense for the domains that do deploy it.

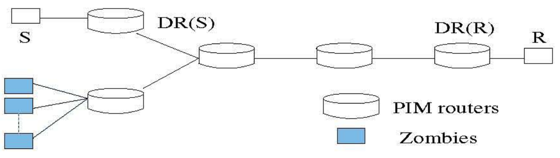


Fig. 6. Simulated network topology.

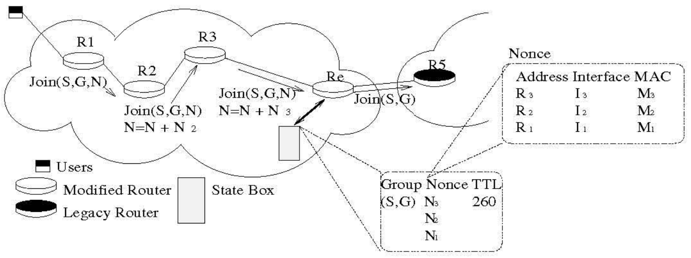


Fig. 7. Partial deployment case.

References

[1] K. Almeroth, "The evolution of multicast: From the MBone to inter-domain multicast to Internet2 deployment," *IEEE Network*, vol. 14, pp. 10–20, January/February 2000.

[2] S. Shenker and J. Wroclawski, "General Characterization Parameters for Integrated Service Network." Internet Engineering Task Force (IETF), RFC 2215, September 1997.

[3] B. Krishnamurthy and C. Wills, "On the use and performance of content distribution networks," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, (San Fransisco, USA), November 2001.

[4] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for IP traceback," in *Proceedings of ACM SIGCOMM*, (Stockholm, SWEDEN), August 2000.

[5] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the IP multicast service and architecture," *IEEE Network*, vol. 14, pp. 78–88, January/February 2000.

[6] P. Savola, R. Lethonen, and D. Meyer, "PIM-SM Multicast Routing Security Issues and Enhancements," October 2004. Internet Engineering Task Force (IETF) draft, work in progress.

[7] D. Estrin et al., "Protocol Independent Multicast Sparse-Mode (PIM-SM): Protocol Specification." Internet Engineering Task Force (IETF), RFC 2362, June 1998.

[8] M. Handley and A. Greenhalgh, "Steps towards a DoS-resistant Internet Architecture," in *Proceedings of ACM SIGCOMM Workshop on Future Directions in Network Architecture*, (Portland, OR, USA), August 2004.

[9] K. Ramachandran and K. Almeroth, "MAFIA: A Multicast Management Solution for Access Control and Packet Filtering," in *Proceedings of MMNS*, (Belfast, IRELAND), September 2003.

[10] R. Lehtonen, J. Soini, J. Majalainen, and H. Vatiainen, "MCOP Operation for first hop routers," June 2004. Internet Engineering Task Force (IETF) draft, work in progress.

[11] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-hashing for message authentication." Internet Engineering Task Force (IETF), RFC 2104, February 1997.