JAM: Mitigating Jellyfish Attacks in Wireless Ad Hoc Networks

Fahad Samad[1,3], Qassem Abu Ahmed[1], Asadullah Shaikh[2,3], and Abdul Aziz[3]

[1] RWTH Aachen University, Germany
samad@i4.de,
qassemabuahmed@gmail.com
[2] Kulliyyah of Information and Communication Technology,
International Islamic University, Malaysia
asadullah.shaikh@live.iium.edu.my
[3] Institute of Business and Technology, Pakistan
abdul.aziz@biztekian.com

Abstract. In recent years, wireless ad hoc networks (WANETs) have become very popular due to their wide range of applications and their ability to be deployed under normal and harsh conditions while supporting high data rates. Although many intrusion detection and trust-based systems have been developed to protect ad hoc networks against misbehaviors such as rushing attacks, query-flood attacks, and selfishness of nodes, these defense mechanisms are still not able to detect protocol compliant attacks called *Jellyfish* (JF) attacks. They target *closed-loop* flows such as TCP that are responsive to network conditions like delay and packet losses and can easily partition the network. In this paper, we introduce a security scheme called JAM (Jellyfish Attacks Mitigator) which can be used to detect and mitigate Jellyfish attacks in ad hoc networks.

Keywords: Wireless Ad hoc Networks, Security, Denial of Service, and Jellyfish Attacks.

1 Introduction

Significant progress has been made in securing ad hoc networks by developing secure routing protocols [1, 2] that ensure different security concepts such as authentication and data integrity. Moreover, intrusion detection and trust-based systems [3, 4, 5, 6, 7, 8] have been developed to protect WANETs against misbehaviors such as rushing attack, query-flood attacks, and selfish behaviors. Yet, most of the defense mechanisms are not able to detect a set of protocol compliant attacks called *jellyfish* (JF) attacks [9].

Similar to a jellyfish which is difficult to be detected until after the sting, jellyfish attacks in ad hoc networks are also hard to detect because they conform to all existing protocol specifications. Jellyfish attackers (JF nodes) can severely reduce the goodput of all traversing closed-loop flows to near zero by periodically dropping a small fraction of packets ($JFPeriodicDroppingAttack$), reordering

B.S. Chowdhry et al. (Eds.): IMTIC 2012, CCIS 281, pp. 432–444, 2012.

them (*JFReorderAttack*), or delaying them (*JFDelayAttack*). A JF attack can severely degrade the performance of a network by preventing long-range communication and thus partitioning the network. For further details about the types of jellyfish attacks, please refer to the work of Aad et al. [9] or the appendix section. In this paper, we present a scheme called JAM [10] to mitigate jellyfish attacks. This scheme detects and circumvents malicious nodes and thus successfully establish routes free from malicious nodes.

2 Existing Schemes

Nadeem and Howarth [3] introduce an *anomaly-based intrusion detection system* (ABIDS). They introduce an ABIDS based on chi-square for intrusion detection. However, this technique relies on a Centralized Entity (CE) to detect anomalies which is not practical in WANETs due to the distributed nature of it. Pirzada and McDonald [5] present a *trust-based mechanism* (TBM) for establishing and managing trust in pure ad-hoc networks where no CE exists and the nodes are not required to be pre-configured. The main drawback of their approach is that the detected malicious nodes are not penalized and isolated from the network. Zouridaki *et al.* [7] introduce robust cooperative trust establishment schemes in Mobile Ad hoc Networks (MANETs) called *E-Hermes*. The scheme can successfully but slowly identify malicious nodes. Moreover, the destination must acknowledge (ACK) each packet and intermediate nodes must set timers. They incur a large overhead by sending a *message authentication* code (MAC) for each intermediate node on a route to assure packets' integrity. De Rango and Marano [6] improve the usage of a well-known secure AODV routing protocol (SAODV) [1] in terms of reducing the number of applied signatures and their verification by intermediate nodes in order to achieve a longer network life.

All the above mentioned trust models are not suitable for detecting jellyfish attacks due to the nature of TCP flows that are responsive to network conditions such as delay and packet loss. A JF node attacks TCP flows for a short time to force them to enter the slow-start state, during which only one segment is sent and the RTO is too high. In such cases, the above mentioned TBMs fail because they would not penalize a node that drops only one packet every few seconds.

3 Proposed Detection Model

There are some notations which have been used to describe our model. Some of the commonly used notations are formulated in Table 1. In our proposed model, a general wireless ad-hoc network is considered. MAC layer acknowledgments are sent by a destination node to notify the source that the sent frame has been successfully received. When a MAC ACK is not received, the source has to resend the unacknowledged frame. Moreover, nodes are considered to operate in promiscuous mode as well. Additionally, a secure AODV protocol such as SAODV for authentication and message integrity is supposed to be working. As we consider intermediate nodes to be attackers, source and destination nodes are assumed to be trusted.

Table 1. Notations used in JAM

Symbol	Definition
$RREQ$	Route request packet
$RREP$	Route reply packet
$RERR$	Route error packet
RTO	Retransmission timeout
CHP	Catalyst helper packets
GPT_{low}	Threshold goodput of a flow lesser than which CHPs are sent
T_h^1	Threshold RTO value of a TCP connection above which CHPs are sent
T_h^2	An RTO value greater than T_h^1
T_{gt}	Time period in which average goodput of a flow is observed
T_{pd}	Observing nodes check for JF attacks after every T_{pd} seconds
T_{chp}^w	Duration for which no CHPs are sent on a route after RREQ message
MIN_INVS	Minimum Threshold for repeatedly detected drop intervals
NUM_FWD	Threshold number of evidences between specific intervals
Q_{max}	Maximum queues managed by observer nodes to log packet forwarding actions and reception time
E_{TH}	Threshold value of overall evidences required to be collected
t_{ex}	The catalyst-helper packets (CHPs) are sent periodically after $t_{ex} + \delta t$ seconds

3.1 Detecting Jellyfish Attacks

The main difficulty in detecting JF attacks consists of the non-continuous misbehavior of attackers. That is, an attacker quickly forces all victim TCP flows to enter the slow-start state so that a few packets will be sent over long periods. This makes it difficult for observing nodes to distinguish between malicious behaviors (packet dropping, reordering, and delaying) and benign behaviors due to network events such as congestion and route change. Therefore, in our proposed model, the TCP protocol is modified so that when it faces a very low goodput or high RTO values, it starts sending packets called *catalyst-helper packets* (CHPs) in a constant ratio to check if a congestion is still there or not. This avoids long waiting times if there is no longer network congestion and allows observing nodes to detect misbehaviors by attackers and hence those nodes can be isolated.

To identify packets in the network, they are supplied with cumulative sequence numbers (SEQs) in clear text. That is, when the routing agent receives a segment from layer 4, the SEQ field of the packet is incremented by one. Moreover, each new flow is supplied with a unique id number (*flow id*). This allows observing nodes to identify packets by 3-tuple values (IP address, flow id, SEQ) and to reduce the amount of bytes needed to store information about each observed packet by only storing these 3-tuple values. Moreover, observing nodes are easily able to detect JFreorder attacks by comparing the SEQs of outgoing packets only.

In order to detect JFperiodic attacks, observing nodes store the reception time of each packet at observed nodes. A packet that is not forwarded within a specific period is considered as dropped. Observing nodes also collect a set of distances between two successive observed drop intervals to emulate the malicious periodic drop interval. When many forwarded packets are observed, the set of offsets relative to the set of distances is determined and the biggest gap is computed.

When the found gap contains several drop intervals within it, a JFperiodic attack is detected. The accuracy of detection improves with an increase in the number of forwarded packets considered. The next subsections will describe the detection scheme of JFperiodic and JFreorder attacks in detail. The detection of JFdelay attacks is beyond the scope of this paper.

Modification of TCP Protocol. TCP congestion control has two main limitations during the slow-start state. First, the nodes back-off exponentially. Second, only one segment is sent between two successive retransmission timeouts (congestion window $cwnd = 1$). For instance, in severe cases it sends one segment and waits for 32 seconds, it then sends another segment and waits for 64 seconds, and then the connection terminates. Backing off exponentially and waiting for longer periods is not necessary in most cases. Moreover, sending a few segments during back off time does not bring the whole network down (unless the whole network is congested). Furthermore, when a relay node in a WANET is congested for a long time, it is likely that a link-breakage event would occur and a router error message RERR is sent to the source node. Therefore, there is a need to modify the TCP protocol to get rid of the above mentioned limitations of the protocol. Yet, the modification should not be in the core functionality of TCP in order to make it adaptable to the existing, wide-spread TCP variants.

We denote the last sent segment with the lowest sequence number that is not yet acknowledged by a catalyst-helper packet (CHP). Sending one CHP during back off time can avoid long waiting times. When this CHP is acknowledged, the RTO is reset to the minimal RTO and the congestion window is doubled. If the congested relay node is freed or another route is established, then the TCP flow resumes sending of the packets.

As depicted in Figure 1, the TCP protocol is modified to send CHPs periodically with a small variation of every $t_{ex} + \delta t$ seconds in the following cases:

- As long as the RTO value of a TCP connection is greater than a threshold value T_h^1 and there exists an established path to the destination.
- The average goodput (in $Kbps$) during the last T_{gt} seconds (GPT_{gt}) is too low, that is, less than a threshold value GPT_{low}.

Moreover, if the RTO value is higher than a threshold value $T_h^2 > T_h^1$, which means that it is high enough to indicate a severe congestion/attack, the source node will initiate the route discovery procedure by sending a route request (RREQ) packet towards the destination node. Additionally, the sending time T_{RQ}^s is stored and no other CHPs on the same route will be sent for T_{chp}^w seconds. High values of T_h^1 and T_{gt} slow down the detection speed of malicious nodes while small values result in sending unnecessary CHPs which increases the congestion in the network if there is no attack. If the RTO is too high, then it is very likely that the route to the destination contains a congested node. Therefore, the source node will try to establish another route by initiating the route discovery procedure. The modification can be incorporated in existing TCP protocols via a patch.

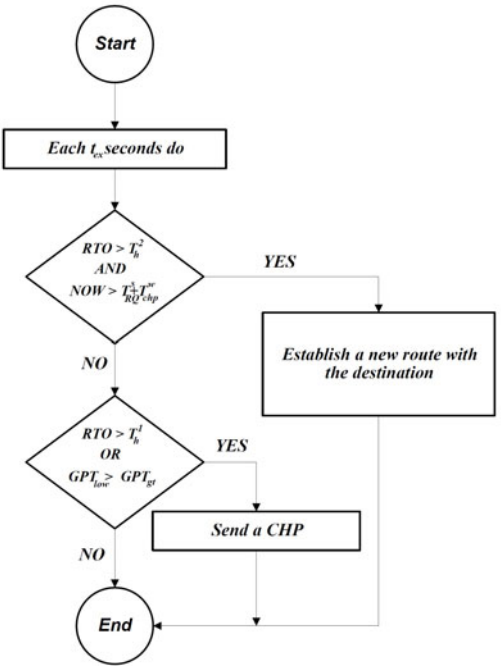

Fig. 1. TCP modification. NOW = current time.

We illustrate the effect of TCP-modification with an example. Consider Figure 2 that describes an experiment conducted on a three-node chain with one relay node and one TCP connection. The AODV routing protocol and the IEEE 802.11 MAC protocol at 11 Mb/sec are used. To achieve the null (zero) goodput, the malicious node (relay node) drops all received packets for 22.5 mili-seconds after every 250 mili-seconds and forwards the rest. The JFperiodic attack starts from the 10-*th* second and the simulation duration is 100 seconds. The values of the modified TCP parameters are: $t_{ex} = 1sec$, $T_h^1 = 3sec$, $T_{gt} = 5sec$, $GPT_{low} = 10Kbps$, and $T_{chp}^w = 5sec$.

When no CHPs are sent, the attacker achieves a successful attack and brings the goodput of the victim flow to null (red line in the figure). All the retransmitted segments have been dropped by the attacker and the RTO timer doubles each time when a retransmission timeout occurs. The second graph (green line) describes the goodput when only one CHP is sent. At 15 sec, when the RTO is equal to 4, the sender sends a CHP which is successfully acknowledged. This causes the RTO value to be reset to the minimal RTO value (0.2 sec) and the congestion window to be doubled. The attack forces the TCP agent to have small RTO- and congestion window values which saves longer waiting time and thus results in a goodput of 50 Kbps. This is equal to 20% of the peak value. This allows nodes to detect attackers by observing a persistent dropping/reordering patterns. During the simulation, only one CHP is sent which incurs a very small overhead.

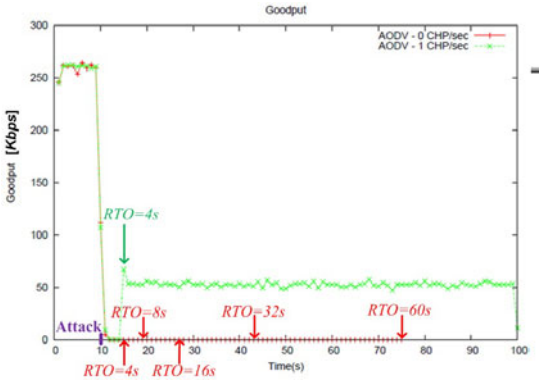

Fig. 2. Impact of sending CHPs on Goodput

In case of severe JFperiodic and JFreorder attacks, the sender needs to send more CHPs per second to accelerate the detection and isolation processes and thus establishes a new route free from malicious nodes.

TCP modification on client side is only a means to help network nodes to detect malicious nodes.

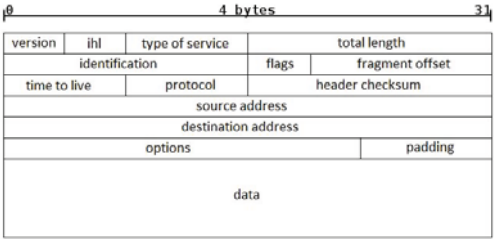

Fig. 3. Structure of the IP Header

Additional Information for IP Packets. Figure 3 describes the structure of the IP header. Segments that are received by the network layer from the TCP agent are supplied with cumulative 16-bit sequence numbers (SEQs) starting from a random number. Moreover, each flow is identified with a unique 16-bit number (*flow id*). A 3-tuple value (IP address, flow id, SEQ) is used to identify IP packets in the network. The uniqueness is guaranteed because no two nodes exist that share the same IP. Each new flow gets a unique flow id by the network layer. Moreover, the flow id and the SEQs are sent in clear text to allow network nodes to identify the packets. Flow ids and SEQs can be incorporated in IP packets by exploiting the option field of IP packets.

Additional Information for Link Layer Acknowledgments. As the name implies, link layer ACKs are small frames that contain only link layer information. That is, observing nodes would not be able to detect when and which

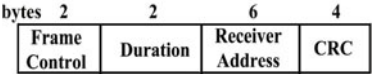

Fig. 4. IEEE 802.11 MAC ACK

packet an observed node has received. Moreover, in some MAC protocols such as the famous IEEE 802.11 MAC protocol, it is not possible for observing nodes to know the originator of a link layer ACK due to the missing ACK-sender address as shown in Figure 4.

Therefore, link layer ACKs are modified to include the following information:

- MAC address of the acknowledging node,
- Source address of the acknowledged packet,
- Flow id of the acknowledged packet, and
- SEQ field of the acknowledged packet.

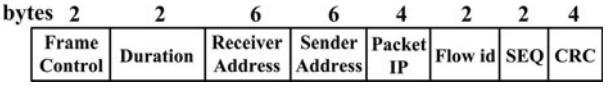

Fig. 5. Modified IEEE 802.11 MAC ACK

After modification, IEEE 802.11 ACKs would look as shown in Figure 5. In this way, each packet's reception and forwarding can be detected and identified by all one-hop nodes. We illustrate it here with an example. As shown in Figure 6, nodes A, C, D, and E are located within the transmission range of node B. When B receives a packet from node A (step 1), B sends a link layer ACK to node A (step 2). Nodes C and E can identify the packet received by B and store the reception time. In step 3, node B forwards the packet to node C. Nodes A, D, and E can see the forwarding action of the same packet. If node B is a JF attacker, it will be detected and isolated simultaneously by all of its one-hop neighbors. When B tries to establish a new route through it by forwarding route request/reply (RREQ/RREP) packets, one-hop neighbors will refuse to process them by dropping them.

One advantage of the new IP and MAC ACK formats is the reduction of the amount of buffer space needed by observing nodes to store information about observed packets. Observing nodes in traditional trust based systems [4,5,6,7,8] store each observed received packet for t seconds. If the observed node does not forward the packet within t seconds, then the packet is considered dropped. If for example 500 packets of size 500 bytes are observed within $t = 2sec$, the maximal amount of buffer space needed is then 250 Kbytes, whereas the amount of buffer space needed in our scheme is very small. For example, an observing node needs to store the IP, flow id, and the first seen SEQ of a flow only once and then stores the SEQ offsets of observed packets relative to the first seen

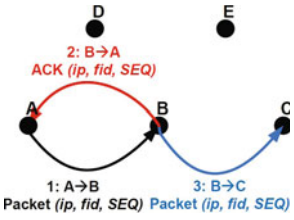

Fig. 6. Detecting actions of nodes

SEQ. However, a question arises that how would node E identify that node B has received a packet if the packet identification- and the MAC sender-fields are unknown? Another problem arises if nodes collect *second-hand-information* (recommendations) about each other: How to detect spurious recommendations?

3.2 Detecting Jellyfish Periodic Dropping Attack

The difficulty in detecting JFperiodic attacks is in differentiating between malicious dropping and dropping due to network events such as congestion. We see this problem from another aspect. If packets are dropped in periodic intervals, then no packets are forwarded in these intervals and all packet forwarding actions occur in other periodic intervals as shown in Figure 7.

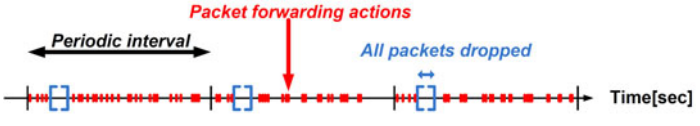

Fig. 7. A JFperiodic Attack

The detection scheme exploits two features. First, packet reception time is independent of the receiver, that is, a node cannot control when to receive packets. Second, packet forwarding actions occur periodically in constant intervals.

The first feature is routing protocol dependent. When a small number of packet retransmission-attempts fail, a link-breakage error occurs and a RERR packet is sent towards source nodes, which is not desired by JF attackers. The second feature is used to distinguish between malicious packet dropping and packet dropping due to network events.

(i) **Logging Actions of Neighbors**
Each node in the network keeps track of its neighbors' actions on information packets. Recall that packets are identified by a 3-tuple value (IP address, flow id, SEQ). We define the possible actions that can be performed on a packet with a specific SEQ number as follows:

- NOT_SEEN: Packet has not been observed by an observing node (default value).
- RECEIVED: Observed node has received the packet, but has not yet forwarded it and the observer has started a RECEIVE_TIMEOUT timer.
- DROPPED: Packet had been received by an observed node and no forwarding action has been observed by the observer till the RECEIVE_TIMEOUT expires.
- FORWARDED: Observed node has forwarded the packet within RECEIVE_TIMEOUT seconds.

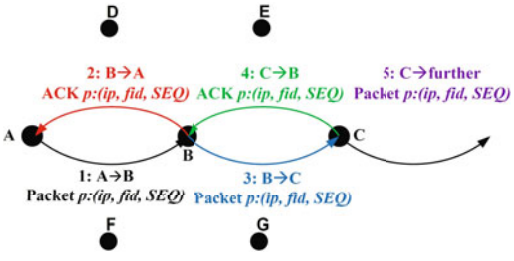

Fig. 8. An Example of Logging of Actions

Logging neighbors' actions is illustrated in an example in Figure 8. Consider that $A \rightarrow B$ means sent from node A to B, $B \rightarrow C$ means sent from node B to C, and so on. Let all nodes be relay nodes and direct neighbors of each other. Furthermore, we denote the action performed by node o on packet p and seen by node n by $act_o^n(p)$. Consider only the updating of node $B's$ actions by nodes D, E, F, and G. Moreover, assume that node D has seen action 2, node E has only seen action 3, node F has only seen action 4, and node G has only seen action 5. When node D observes action 2, it sets $act_B^D(p)$ =RECEIVED and after RECEIVE_TIMEOUT seconds it sets $act_B^D(p)$ =DROPPED. When node E observes action 3, it sets $act_B^E(p)$ =FORWARDED. When node F observes action 4, it sets $act_B^F(p)$ =FORWARDED. When node G observes action 5, it sets $act_B^G(p)$ =FORWARDED.

As mentioned earlier, we only concentrate on JF attacks that target the whole network. Therefore, each observing node n manages a queue Q_o^n for all flows that traverse an observed neighbor o. Queue Q_o^n contains the logged actions together with the reception time of each logged packet (monotonically increasing) that has been observed until RECEIVE_TIMEOUT expires. Moreover, Q_o^n has a maximum size of Q_{max}, which plays an important role in the speed of detecting malicious nodes and it will be discussed in the next subsections. An entry of the queue Q_o^n consists of the following tuple:

Reception time of packet p	$action \in \{DROP, FWD\}$

(ii) **Collecting Candidate Periodic Intervals.** Congestions and thus packet dropping actions are inevitable in WANETs. Therefore, JFperiodic attackers might also drop packets during the forwarding phase due to congestion. This makes it harder for observing nodes to detect constant periodic intervals that consist of pure packet forwarding and pure packet dropping subintervals as shown in Figure 7. We transform the queue Q_o^n into a list of intervals denoted by ***ltr*** that consists of alternating ***pure*** drop- and forward-intervals. That is, each two successive equal actions belong to the same interval. Both of the successive equal actions are either in drop or in forward intervals. Each interval x has the following parameters:

- ***x.stime***: start time of interval x,
- ***x.etime***: end time of interval x,
- ***x.middle***: middle time of interval x.

The set of candidate periodic intervals ($DISTANCES$) is then collected. For each drop interval x in *ltr* and for each drop interval y in *ltr* which is close to x, i.e. $|y.stime - x.stime| < T_{inv}$, where T_{inv} is a threshold value that expresses the distance between the intervals x and y, the differences between the start-, middle- and the end times of x and y are inserted into $DISTANCES$. The flow chart in Figure 9 illustrates how the set $DISTANCES$ is formed. For small values of T_{inv} (T_{inv} is smaller than the periodic drop interval length), it is very likely that the JFperiodic attack is not detected, while high values of T_{inv} may lead to unnecessary computational overhead.

(iii) **Detection Mechanism**

The detection mechanism is based on collecting *FORWARDED* actions, denoted by *evidences*. The more evidences are collected, the more accurate will be the detection of JFperiodic attacks. The detection mechanism is run for each *distance* in $DISTANCES$.

When the number of evidences is greater than a threshold value E_{TH}, the following steps are carried out:

(a) The time axis starting from the oldest action in Q_o^n till the most recent action in Q_o^n is divided into intervals of length *distance* (Figure 10 a).
(b) A new empty interval of length *distance* is created, denoted by INV.
(c) The offset of each evidence relative to the start time of its interval is computed (Figure 10 b).
(d) Each computed offset is then added to INV (Figure 10 c).
(e) The maximum gap g_1 between two successive evidences in INV is determined.
(f) Let g_2 be the difference between *distance*+the lowest offset in INV and the highest offset in INV. The maximum gap g is then equal to the maximum between g_1 and g_2.

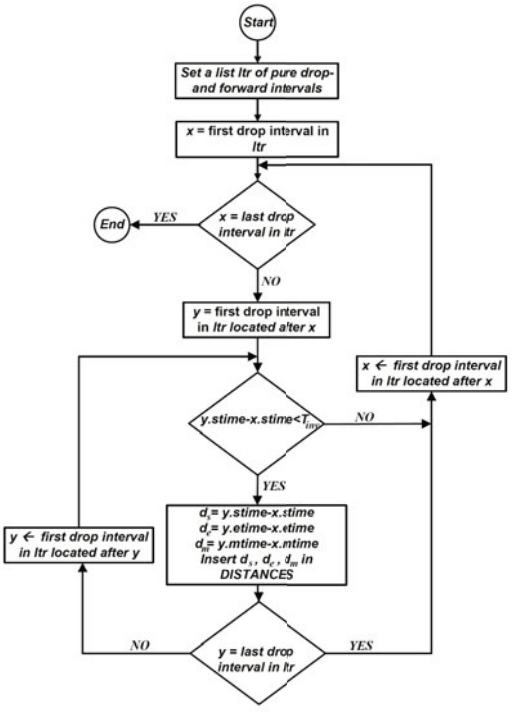

Fig. 9. Setting value of $DISTANCES$

(g) Let num be the number of drop intervals in ltr that are smaller than the gap g and are really located inside the found gap (offsets compared). If num is greater than a threshold value MIN_NUM, then a JFperiodic attack is detected.

The higher the E_{TH}, the more accurate the detection scheme. If the observed node o is benign, then the evidences in INV will be distributed throughout the whole interval and the maximum gap g will be very small so that it will not contain any drop intervals. If node o is malicious, the value of g will be high enough to include several pure drop intervals within it. Empirical results show that when $E_{TH} > 300 \cdot distance$, there will be no false positives. This means that there will be no benign nodes that are detected as JFperiodic attackers. Lower values might consider one or two benign nodes as attackers.

Q_{max}, the maximum size of Q_o^n, plays an important role in the detection speed of JFperiodic attackers. High values of Q_{max} achieve more accuracy in the detection mechanism. On the other hand, the detection speed gets slower because the queue Q_o^n would contain old actions that have been observed before the attack has started. These actions will be considered

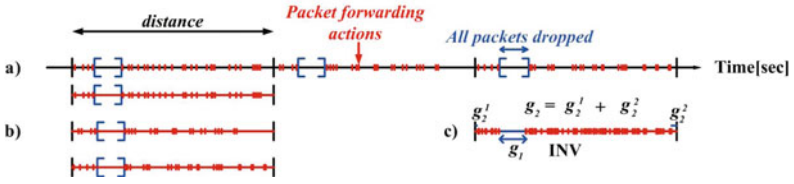

Fig. 10. Detecting JFperiodic Attacks

in the detection algorithm and will therefore result in smaller values of gap g and num. Hence, the probability to detect the attacks gets lower.

3.3 Detecting Jellyfish Reorder Attack

Including flow ids and cumulative SEQ numbers in IP packets makes the detection of JFreorder attacks very easy. This is because observing nodes can detect persistent reordering of packets by comparing the SEQ numbers of outgoing packets (of the same flow). If an observed node sends packets with SEQ number lower than the maximum sent one, a misbehavior is detected. If misbehaviors continue for at least $T_{reorder}$ seconds, a JFreorder attack is detected. If a victim flow suffers from a very low goodput, it starts sending CHPs and the attacker continues reordering packets and thus the attack will be detected by one-hop neighbors. Packet reordering events in WANETs are rare, short-lived, and occur due to network events such as route changes [9]. Threshold time $T_{reorder}$ must be longer than the duration of packet reordering due to these network events.

4 Conclusion

In this research paper a mitigation scheme to detect jellyfish attacks has been introduced. Future work includes the development of a new security scheme to detect and isolate colluding nodes, to enable source nodes to detect jellyfish nodes, and to exclude them from being part of new routes.

References

1. Zapata, M.G.: Secure ad hoc on-demand distance vector (saodv) routing. INTERNET-DRAFT draft-guerreromanet-saodv-06 (September 2006)
2. Kargl, et al.: Secure dynamic source routing. In: Hawaii International Conference on System Sciences, vol. 9, p. 320c (2005)
3. Nadeem, A., Howarth, M.: Adaptive intrusion detection & prevention of denial of service attacks in manets. In: Proceedings of the Int. Conf. on Wireless Comm. and Mobile Computing: Connecting the World Wirelessly, IWCMC 2009, pp. 926–930 (2009)
4. Boukerche, et al.: An adaptive computational trust model for mobile ad hoc networks. In: Proceedings of Int. Conf. on Wireless Comm. and Mobile Computing: Connecting the World Wirelessly, IWCMC 2009, pp. 191–195 (2009)

5. Pirzada, A., Mcdonald, C.: Trust establishment in pure ad-hoc networks. Wireless Personal Communications 37, 139–168 (2006)
6. De Rango, F., Marano, S.: Trust-based saodv protocol with intrusion detection and incentive cooperation in manet. In: Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly, IWCMC 2009, pp. 1443–1448. ACM (2009)
7. Zouridaki, C., Mark, B.L.: Robust cooperative trust establishment for manets. In: Proceedings of the Fourth ACM Workshop on Security of ad Hoc and Sensor Networks, SASN 2006, pp. 23–34. ACM (2006)
8. Jiang, et al.: A scalable and robust approach to collaboration enforcement in mobile ad-hoc networks. Communication and Networks 9(1), 56–66 (2007)
9. Aad, et al.: Impact of denial of service attacks on ad hoc networks. IEEE/ACM Trans. Netw. 16, 791–802 (2008)
10. Samad, F.: Securing Wireless Mesh Networks - A Three Dimensional Perspective. PhD thesis, RWTH Aachen University, Germany (2011)