Detecting Malicious Packet Losses

Alper T. Mızrak, Keith Marzullo, and Stefan Savage

Abstract

In this paper we consider the problem of detecting whether a compromised router is maliciously manipulating its stream of packets. In particular, we are concerned with a simple yet effective attack in which a router selectively drops packets destined for some victim. Unfortunately, it is quite challenging to attribute a missing packet to a malicious action because normal network congestion can produce the same effect. Modern networks routinely drop packets when the load temporarily exceeds a router's buffering capacity. Previous detection protocols have tried to address this problem using a user-defined threshold: too many dropped packets implies malicious intent. However this heuristic is fundamentally unsound; setting this threshold is, at best, an art and will necessarily create unnecessary false positives or mask highly-focused attacks.

We have designed, developed and implemented a compromised router detection protocol that dynamically infers, based on measured traffic rates and buffer sizes, the number of congestive packet losses that will occur. Once the ambiguity from congestion is removed, subsequent packet losses can be attributed to malicious actions. We have tested our protocol in Emulab and have studied its effectiveness in differentiating attacks from legitimate network behavior.

Index Terms

Internet dependability, Intrusion detection and tolerance, Distributed systems, Reliable networks, Malicious routers.

I. Introduction

The Internet is not a safe place. Unsecured hosts can expect to be compromised within minutes of connecting to the Internet and even well-protected hosts may be crippled with denial-of-service attacks. However, while such threats to host systems are widely understood, it is less well appreciated that the network infrastructure itself is subject to constant attack as well. Indeed, through combinations of social engineering and weak passwords, attackers have seized control over of thousands of Internet routers [1], [2]. Even more troubling is Mike Lynn's controversial presentation at the 2005 Black Hat Briefings, which demonstrated how Cisco routers can be compromised via simple software vulnerabilities. Once a router has been compromised in such a fashion, an attacker may interpose on the traffic stream and manipulate it maliciously to attack others – selectively dropping, modifying, or re-routing packets.

Several researchers have developed distributed protocols to detect such traffic manipulations, typically by validating that traffic transmitted by one router is received un-modified by another [3], [4]. However, all of these schemes – including our own – struggle in interpreting the *absence* of traffic. While a packet that has been modified in transit represents clear evidence of tampering, a missing packet is inherently ambiguous: it may have been explicitly blocked by a compromised router or it may have been dropped benignly due to network congestion. In fact, modern routers routinely drop packets due to bursts in traffic that exceed a router's buffering capacity, and the widely-used Transmission Control Protocol (TCP) is designed to *cause* such losses as part of its normal congestion control behavior. Thus, existing traffic validation systems must inevitably produce false positives for benign events and/or produce false negatives by failing to report real malicious packet dropping.

In this paper, we develop a compromised router detection protocol that dynamically infers the precise number of congestive packet losses that will occur. Once the congestion ambiguity is removed, subsequent packet losses can be safely attributed to malicious actions. We believe our protocol is the first to automatically predict congestion in a systematic manner and is necessary for making any such network fault detection practical.

The authors are with the University of California, San Diego.
Email: {amizrak, marzullo, savage}@cs.ucsd.edu
Mail: Dept. of Computer Science and Engineering, UCSD
9500 Gilman Drive, EBU 3B
La Jolla, CA 92093-0404

In the remainder of this paper we briefly survey the related background material, evaluate options for inferring congestion, and then present the assumptions, specification and a formal description of a protocol that achieves these goals. We have evaluated our protocol in a small experimental network and demonstrate that it is capable of accurately resolving extremely small and fine-grained attacks.

II. Background

There are inherently two threats posed by a compromised router. The attacker may subvert the network control plane (e.g. by manipulating the routing protocol into false route updates) or may subvert the network data plane and forward individual packets incorrectly. The first set of attacks have seen the widest interest and the most activity – largely due to their catastrophic potential. By violating the routing protocol itself and attacker may cause large portions of the network to become inoperable. Thus, there have been a variety of efforts to impart authenticity and consistency guarantees on route update messages with varying levels of cost and protection [5], [6], [7], [8], [9], [10]. We do not consider this class of attacks in this paper.

Instead, we have focused on the less well appreciated threat of an attacker subverting the packet forwarding process on a compromised router. Such an attack presents a wide set of opportunities including denial-of-service, surveillance, man-in-the-middle attacks, replay and insertion attacks, and so on. Moreover, most of these attacks can be trivially implemented via the existing command shell languages in commodity routers.

The earliest work on fault-tolerant forwarding is due to Perlman [11] who developed a robust routing system based on source routing, digitally signed *route-setup packets*, reserved buffers. While groundbreaking, Perlman's work required significant commitments of router resources and high levels of network participation to detect anomalies. Since, a variety of researchers have proposed lighter-weight protocols for *actively probing* the network to test whether packets are forwarded in a manner consistent with the advertised global topology [5], [12], [13]. Conversely, the 1997 WATCHERS system detects disruptive routers passively via a distributed monitoring algorithm that detects deviations from a "conservation of flow" invariant [14], [3]. However, work on WATCHERS was abandoned, in part due to limitations in its distributed detection protocol, its overhead, and the problem of ambiguity stemming from congestion [15]. Finally, our own work broke the problem into three pieces: a traffic validation mechanism, a distributed detection protocol, and a re-routing countermeasure. In [16], [4] we focused on the detection protocol, provided a formal framework for evaluating the accuracy and precision of any such protocol and described several practical protocols that allow scalable implementations. However, we also assumed that the problem of congestion ambiguity could be solved, without providing a solution. This paper presents a protocol that removes this assumption.

III. Inferring Congestive Loss

In building a traffic validation protocol, it is necessary to explicitly resolve the ambiguity around packet losses. Should the absence of a given packet be seen as malicious or benign? In practice there are three approaches for addressing this issue:

- *Static Threshold*. Low rates of packet loss are assumed to be congestive, while rates above some predefined threshold are deemed malicious.
- *Traffic modeling*. Packet loss rates are predicted as a function of traffic parameters, losses beyond the prediction are deemed malicious.
- *Traffic measurement*. Individual packet losses are predicted as a function of measured traffic load and router buffer capacity. Deviations from these predictions are deemed malicious.

Most traffic validation protocols, including `WATCHERS` [3], Secure Traceroute [12] and our own work described in [4], analyze aggregate traffic over some period of time in order to amortize monitoring overhead over many packets. For example, one validation protocol described in [4] maintains packet counters in each router to detect if traffic flow is not conserved from source to destination. When a packet arrives at router r and is forwarded to a destination that will traverse a path segment ending at router x, r increments an outbound counter associated with router x. Conversely, when the packet arrives at router r it increments an inbound counter associated with router x. Periodically, router x sends a copy of its outbound counters to the associated routers for validation. Then a given router r can compare the number of packets which x claims to have sent to it with the number of packets it count as being received from there, and can detect the number of packet losses.

Thus, over some time window a router simply knows that out of m packets sent, n were successfully received. To address congestion ambiguity, all of these systems employ a pre-defined threshold: if more than this number are dropped in a time interval, then one assumes that some router is compromised. However, this heuristic is fundamentally flawed: how does one choose the threshold?

In order to avoid false positives, the threshold must be large enough to include the maximum number of possible congestive legitimate packet losses over a measurement interval. Thus, any compromised router can drop that many packets without being detected. Unfortunately, given the nature of the dominant Transmission Control Protocol, even small numbers of losses can have significant impacts. Subtle attackers can selectively target the traffic flows of a single victim and within these flows only drop those packets that cause the most harm. For example, losing a TCP SYN packet used in connection establishment has a disproportionate impact on a host because the retransmission timeout must necessarily be very long (typically 3 seconds or more). Other seemingly minor attacks that cause TCP timeouts can have similar effects – a class of attacks well described in [17].

Instead of using a static threshold, if the probability of congestive losses can be modeled then one could resolve ambiguities by comparing measured loss rates to the rates predicted by the model. On approach for doing this is to predict congestion analytically as a function of individual traffic flow parameters, since TCP explicitly responds to congestion. Indeed, the behavior of TCP has been excessively studied [18], [19], [20], [21], [22]. A simplified[1] stochastic model of TCP congestion control yields the following famous square root formula:

$$B = \frac{1}{RTT}\sqrt{\frac{3}{2bp}}$$

where B is the throughput of the connection, RTT is the average round trip time, b is the number of packets that are acknowledged by one ACK, and p is the probability that a TCP packet is lost. The steady-state throughput of long-lived TCP flows can be described by this formula as a function of RTT and p.

This formula is based on a constant loss probability, which is the simplest model, but others have extended this work to encompass a variety of loss processes [22], [20], [23], [24]. None of these have been able to capture congestion behavior in all situations.

Another approach is to model congestion for the aggregate capacity of a link. In [25], Appenzeller *et.al.* explore the question of "How much buffering do routers need?". A widely applied rule-of-thumb suggests that router must be able to buffer a full delay bandwidth product. This controversial paper argues that due to congestion control effects, the rule-of-thumb is wrong, and the amount of required buffering is proportional to the square root of the total number of TCP flows. To achieve this, the authors produced an analytic model of buffer occupancy as a function of TCP behavior. We have evaluated their model thoroughly and have communicated with the authors, who agree that their model is only a rough approximation that ignores many details of TCP, including timeouts, residual synchronization, and many other effects. Thus, while the analysis is robust enough to model buffer size it is not precise enough to predict congestive loss accurately.

Hence, we have turned to measuring the interaction of traffic load and buffer occupancy explicitly. Given an output buffered FIFO router, congestion can be predicted precisely as a function of the inputs (the traffic rate delivered from all input ports destined to the target output port), the capacity of the output buffer, and the speed of the output link. A packet will be lost only if packet input rates from all sources exceed the output link speed for long enough. If such measurements are taken with high precision it should even be possible to predict individual packet losses. It is this approach that we consider further in the reset of this paper. We restrict our discussion to output buffered switches for simplicity although the same approach can be extended to input buffered switches or virtual output queues with additional adjustments (and overhead).

Because of some uncertainty in the system, we can not predict exactly which individual packets will be dropped. So, our approach is still based on thresholds. Instead of being a threshold on rate, it is a threshold on a statistical measure: the amount of confidence that the drop was due to a malicious attack rather than from some normal router function. To make this distinction clearer, we refer to the statistical threshold as the *target significance value*.

IV. System Model

In this section, we briefly describe the assumptions underlying our model.

[1]This formula omits many TCP dynamics such as timeouts, slow start, delayed acks, etc. More complex formulas taking these into account can be found in literature.

A. Network Model

We use the same system model as in our earlier work [4]. We consider a network to consist of individual homogeneous routers interconnected via directional point-to-point links. This model is an intentional simplification of real networks (e.g., it does not include broadcast channels or independently failing network interfaces) but is sufficiently general to encompass such details if necessary. Unlike our earlier work, we assume that the bandwidth, and the delay of each link, and the queue limit for each interface are all known publicly.

Within a network, we presume that packets are forwarded in a hop-by-hop fashion, based on a local forwarding table. These forwarding tables are updated via a distributed link-state routing protocol such as OSPF or IS-IS. This is critical, as we depend on the routing protocol to provide each node with a global view of the current network topology. Finally, we assume the administrative ability to assign and distribute cryptographic keys to sets of nearby routers. This overall model is consistent with the typical construction of large enterprise IP networks or the internal structure of single ISP backbone networks, but is not well-suited for networks that are composed of multiple administrative domains using BGP. At this level of abstraction, we can assume a synchronous network model.

We define a *path* to be a finite sequence $\langle r_1, r_2, \ldots r_n \rangle$ of adjacent routers. Operationally, a path defines a sequence of routers a packet can follow. We call the first router of the path the *source* and the last router its *sink*; together, these are called *terminal routers*. A path might consist of only one router, in which case the source and sink are the same. Terminal routers are leaf routers: they are never in the middle of any path.

An $x-$*path segment* is a consecutive sequence of x routers that is a subsequence of a path. A *path segment* is an $x-$path segment for some value of $x > 0$. For example, if a network consists of the single path $\langle a, b, c, d \rangle$ then $\langle c, d \rangle$ and $\langle b, c \rangle$ are both 2-path segments, but $\langle a, c \rangle$ is not because a and c are not adjacent.

B. Threat Model

As explained in the introduction, this paper focuses solely on data plane attacks (control plane attacks can be addressed by other protocols with appropriate threat models such as, [6], [7], [5], [8], [9], [10]). Moreover, for simplicity we examine only attacks that involve packet dropping. However, our approach is easily extended to address other attacks – such as packet modification or reordering – similar to our previous work. Finally, as in [4], the protocol we develop validates traffic whose source and sink routers are uncompromised.

A router can be *traffic faulty* by maliciously dropping packets and *protocol faulty* by not following the rules of the detection protocol. We say that a compromised router r is *t-faulty* (that is, *traffic faulty*) with respect to a path segment π during τ if π contains r and, during the period of time τ, r maliciously drops or misroutes packets that flow through π. A router can drop packets without being faulty, as long as the packets are dropped because the corresponding output interface is congested. A compromised router r can also behave in an arbitrarily malicious way in terms of executing the protocol we present, in which case we indicate r as *p-faulty* (that is, *protocol faulty*). A p-faulty router can send control messages with arbitrarily faulty information, or it can simply not send some or all of them. A *faulty* router is one that is t-faulty, p-faulty or both.

Attackers can compromise one or more routers in a network. However, for simplicity we assume in this paper that adjacent routers cannot be *faulty*. Our work is easily extended to the case of k adjacent *faulty* routers.

V. PROTOCOL χ

Protocol χ detects *traffic faulty* routers by validating the queue of each output interface for each router. Given the buffer size and the rate traffic enters and exits a queue, the behavior of the queue is deterministic. If the actual behavior deviates from the predicted behavior, then a failure has occurred.

We present the failure detection protocol in terms of the solutions of the distinct subproblems: traffic validation, information dissemination, and response.

A. Traffic Validation

The first problem we address is *traffic validation*: what information is collected about traffic and how it is used to determine that a router has been compromised. We denote with *Tinfo*$(r, \mathbf{Q}_{dir}, \pi, \tau)$. the traffic information collected by router r that traversed path segment π over time interval τ. $\mathbf{Q}_{dir}$ is either Q_{in}, meaning traffic into Q, or Q_{out}, meaning traffic out of Q.

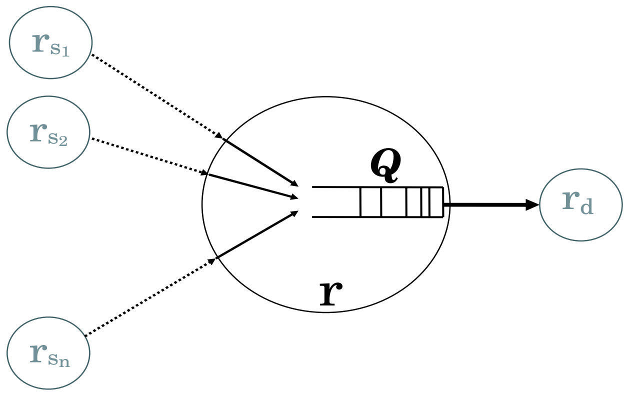


Fig. 1. Validating the queue of an output interface.

$$\begin{aligned}
c_{single} &= Prob(\mathit{fp} \text{ is maliciously dropped}) \\
&= Prob(\text{there is enough space in the queue to buffer } \mathit{fp}) \\
&= Prob(q_{act}(\mathit{ts}) + \mathit{ps} \leq q_{limit}) \\
&= Prob(X + q_{pred}(\mathit{ts}) + \mathit{ps} \leq q_{limit}) && \text{Random variable } X = q_{act}(\mathit{ts}) - q_{pred}(\mathit{ts}) \\
& && \text{with mean } \mu \text{ and standard deviation } \sigma \\
&= Prob(X \leq q_{limit} - q_{pred}(\mathit{ts}) - \mathit{ps}) \\
&= Prob(Y \leq \frac{q_{limit} - q_{pred}(\mathit{ts}) - \mathit{ps} - \mu}{\sigma}) && \text{Random variable } Y = (X - \mu)/\sigma \\
&= Prob(Y \leq y_1) && y_1 = \frac{q_{limit} - q_{pred}(\mathit{ts}) - \mathit{ps} - \mu}{\sigma} \\
&= \frac{1 + \mathit{erf}(y_1/\sqrt{2})}{2} && \mathit{erf} \text{ is the error function.}
\end{aligned}$$

Fig. 2. Confidence value for single packet loss test for a packet with a fingerprint *fp*, size *ps*, and a time stamp *ts*.

Consider the queue Q in a router r associated with the output interface of link $\langle r, r_d \rangle$. See Figure 1. The neighbor routers $r_{s_1}, r_{s_2}, \ldots r_{s_n}$ feed data into Q. At an abstract level, we represent traffic a validation mechanism associated with Q as a predicate $\mathit{TV}(Q, q_{pred}(t), S, D)$ where:

- $q_{pred}(t)$ is the predicted state of Q at time t. $q_{pred}(t)$ is initialized to 0 when the link $\langle r, r_d \rangle$ is discovered and installed into the routing fabric. q_{pred} is updated as part of traffic validation.
- $S = \{\forall i \in \{1, 2, ..., n\} : \mathit{Tinfo}(r_{s_i}, Q_{in}, \langle r_{s_i}, r, r_d \rangle, \tau)\ \}$, is a set of information about traffic coming into Q as collected by neighbor routers.
- $D = \mathit{Tinfo}(r_d, Q_{out}, \langle r, r_d \rangle, \tau)$ is the traffic information about the outgoing traffic from Q collected at router r_d.

If routers r_{s_1}, r_{s_2},..., r_{s_n} and r_d are not *p-faulty*, then $\mathit{TV}(Q, q_{pred}(t), S, D)$ evaluates to *false* iff r was *t-faulty* and dropped packets maliciously during τ.

$\mathit{Tinfo}(r, \mathbf{Q}_{dir}, \pi, \tau)$ can be represented in different ways. We use a set that contains, for each packet traversing Q, a three-tuple that contains: a fingerprint of the packet, the packet's size and the time that the packet entered or exited Q (depending on whether $\mathbf{Q}_{dir}$ is Q_{in} or Q_{out}). For example, if at time t router r_s transmits a packet of size *ps* bytes with a fingerprint *fp*, and the packet is to traverse π, then r_s computes when the packet will enter Q based on the packet's transmission and propagation delay. Given a link delay d and link bandwidth bw associated with the link $\langle r_s, r \rangle$, the time stamp for the packet is $t + d + \mathit{ps}/\mathit{bw}$.

TV can be implemented by simulating the behavior of Q. Let P be a priority queue, sorted by increasing time stamp. All the traffic information S and D are inserted into P along with the identity of the set (S or D), from which the information came. Then, P is enumerated. For each packet in P with a fingerprint *fp*, size *ps*, and a time stamp *ts*, q_{pred} is updated as follows. Assume t is the time stamp of the last packet evaluated before the current one:

- If *fp* came from D, then the packet is leaving Q: $q_{pred}(ts) := q_{pred}(t) - ps$.
- If *fp* came from S and ($fp \in D$), then the packet *fp* is entering and will exit: $q_{pred}(ts) := q_{pred}(t) + ps$.
- If *fp* came from S and ($fp \notin D$), then the packet *fp* is entering into Q and the packet *fp* would not be transmitted in the future: $q_{pred}(ts)$ is unchanged, and the packet is *dropped*.
  - If $q_{limit} < q_{pred}(t) + ps$, where q_{limit} is the buffer limit of Q, then the packet is dropped due to congestion.
  - Otherwise, the packet is dropped due to malicious attack. Detect the failure.

In practice, the behavior of a queue cannot be predicted with complete accuracy. For example, the tuples in S and D may be collected over slightly different intervals, and so a packet may appear to be dropped when in fact it is not (this is discussed in Section VI-A). Or, a packet sent to a router may not enter the queue at the expected time because of short-term scheduling delay and internal processing delays.

Let $q_{act}(t)$ be the actual queue length at time t. Based on the central limit theorem[2] [26], our intuition tells us that the error $q_{error} = q_{act} - q_{pred}$ can be approximated with a normal distribution. Indeed, this turns out to be the case as we show in Section VII. Hence, this suggests using a probabilistic approach.

We use two tests: one based on the loss of a single packet and one based on the loss of a set of packets.

1) Single packet loss test: If a packet with fingerprint *fp* and size *ps* is dropped at time *ts* when the predicted queue length is $q_{pred}(ts)$ then we raise an alarm with a confidence value c_{single}, which is the probability of the packet being dropped maliciously. c_{single} is computed as in Figure 2.

The mean μ and standard deviation σ of X can be determined by monitoring during a learning period. We don't expect μ and σ change much over time, because they are in turn determined by values that themselves don't change much over time. Hence, the learning period need not be done very often.

A malicious router is detected if the confidence value c_{single} is at least a target significance level s_{single}^{level}.[3]

2) Combined packet losses test: The second test is useful when more than one packet is dropped during a round and the first test does not detect a malicious router. It is based on the well-known Z-test[4] [26]. Let L be the set of $n > 1$ packets dropped during the last time interval. For the packets in L, let $\overline{ps}$ be the mean of the packet sizes, $\overline{q_{pred}}$ be the mean of $q_{pred}(ts)$ (the predicted queue length) and $\overline{q_{act}}$ be the mean of $q_{act}(ts)$ (the actual queue length) over the times the packets were dropped.

We test the hypothesis of "The packets are lost due to malicious attack": $\mu > q_{limit} - \overline{q_{pred}} - \overline{ps}$. The Z-test score is:

$$z_1 = \frac{(q_{limit} - \overline{q_{pred}} - \overline{ps} - \mu)}{\sigma\sqrt{n}}$$

For the standard normal distribution Z, the probability of $Prob(Z < z_1)$ gives the confidence value $c_{combined}$ for the hypothesis. A malicious router is detected if $c_{combined}$ is at least a target significance level $s_{combined}^{level}$.

One can question using a Z-test in this way because the set of dropped packets are not a simple random sample. But, this test is used when there are packets being dropped and the first test determined that they were consistent with congestion loss. Hence, the router is under load during the short period the measurement was taken and most of the points, both for dropped packets and for non-dropped packets, should have a nearly-full Q. In Section VII we show that the Z-test does in fact detect a router that is malicious in a calculated manner.

B. Distributed Detection

Since the behavior of the queue is deterministic, the traffic validation mechanisms detects *traffic faulty* routers whenever the actual behavior of the queue deviates from the predicted behavior. However, a faulty router can also be *protocol faulty*: it can behave arbitrarily with respect to the protocol, such as dropping or altering the control messages of χ. We mask the effect of p-faulty routers using distributed detection.

[2]The central limit theorem states the following. Consider a set of n samples drawn independently from any given distribution. As n increases, the average of the samples approaches a normal distribution as long as the sum of the samples has a finite variance.

[3]The significance level is the critical value used to decide to reject the null hypothesis in traditional statistical hypothesis testing. If it is rejected, then the outcome of the experiment said to be statistically significant with that significance level.

[4]The Z-test, which is a statistical test, is used to decide whether the difference between a sample mean and a given population mean is large enough to be statistically significant or not.

Given TV, we need to distribute the necessary traffic information among the routers and implement a distributed detection protocol. Every outbound interface queue Q in the network is monitored by the neighboring routers and validated by a router r_d such that Q is associated with the link $\langle r, r_d \rangle$.

With respect to a given Q, the routers involved in detection are (as shown in Figure 1):

- r_{s_*}, which send traffic into Q to be forwarded.
- r, which hosts Q.
- r_d, which is the router to which Q's outgoing traffic is forwarded.

Each involved router has a different role, described below.

1) Traffic Information Collection: Each router collects the following traffic information during a time interval τ:

- r_{s_*}: Collect $\mathit{Tinfo}(r_{s_*}, Q_{in}, \langle r_{s_*}, r, r_d \rangle, \tau)$.
- r: Collect $\mathit{Tinfo}(r, Q_{in}, \langle r_{s_*}, r, r_d \rangle, \tau)$. This information is used to check the transit traffic information sent by r_{s_*} routers.
- r_d: Collect $\mathit{Tinfo}(r_d, Q_{out}, \langle r, r_d \rangle, \tau)$.

2) Information Dissemination and Detection:

- r_{s_*}: At the end of each time interval τ, router r_{s_*} sends $[\mathit{Tinfo}(r_{s_*}, Q_{in}, \langle r_{s_*}, r, r_d \rangle, \tau)]_{r_{s_*}}$ that it has collected. $[M]_x$ is a message M digitally signed by x. Digital signatures are required for integrity and authenticity against message tampering. [5]

D-I. r: Let Δ be the upper bound on the time to forward traffic information.

a) If r does not receive traffic information from r_{s_*} within Δ, then r detects $\langle r_{s_*}, r \rangle$.

b) Upon receiving $[\mathit{Tinfo}(r_{s_*}, Q_{in}, \langle r_{s_*}, r, r_d \rangle, \tau)]_{r_{s_*}}$ router r verifies the signature and checks to see if this information is equal to its own copy $\mathit{Tinfo}(r, Q_{in}, \langle r_{s_*}, r, r_d \rangle, \tau)$. If so, then r forwards it to r_d. If not, then r detects $\langle r_{s_*}, r \rangle$.

At this point, if r has detected a failure $\langle r_{s_*}, r \rangle$, then it forwards its own copy of traffic information $\mathit{Tinfo}(r, Q_{in}, \langle r_{s_*}, r, r_d \rangle, \tau)$. This is required by r_d to simulate Q's behavior and keep the state q up to date.

D-II. r_d:

a) If r_d does not receive traffic information $\mathit{Tinfo}(r_{s_*}, Q_{in}, \langle r_{s_*}, r, r_d \rangle, \tau)$ originated by r_{s_*} within 2Δ, then it expects r to have detected r_{s_*} as faulty and to announce this detection through the response mechanism. If r does not do this, then r_d detects $\langle r, r_d \rangle$.

b) After receiving the traffic information forwarded from r, r_d checks the integrity and authenticity of the message. If the digital signature verification fails, then r_d detects $\langle r, r_d \rangle$.

c) Collecting all traffic information, router r_d evaluates the TV predicate for queue Q. If TV evaluates to *false*, then r_d detects $\langle r, r_d \rangle$.

Fault detections D-Ia, D-Ib, D-IIa, and D-IIb are due to *p-faulty* routers, and fault detection D-IIc is due to the traffic validation detecting *t-faulty* routers.

Note that dropping traffic information packets due to congestion can lead to false positives. Thus, the routers send this data with high priority. Doing so may cause other data to be dropped instead as congestion. Traffic validation needs to take this into account. It is not hard, but it is somewhat detailed, to do so in simulating $Q's$ behavior.

C. Response

Once a router r detects router r' as faulty, r announces the link $\langle r, r' \rangle$ as being suspected. This this suspicion is disseminated through the distributed link state flooding mechanism of the routing protocol. As a consequence, the suspected link is removed from the routing fabric.

Of course, a p-faulty router r can announce a link $\langle r, r' \rangle$ as being faulty, but it can do this for any routing protocol. And, in doing so, it only stops traffic from being routed through itself. Router r could even do this by simply crashing itself. To protect against such attack, the routing fabric needs to have sufficient path redundancy.

VI. Analysis of Protocol χ

In this section, we consider the properties and overhead of protocol χ.

[5]Digital signatures can be replaced with message authentication codes if the secret keys are distributed among the routers.

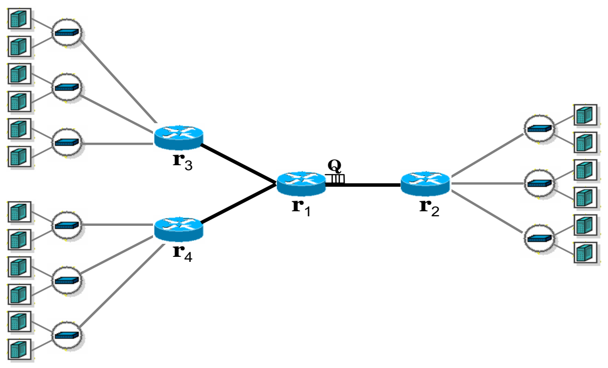


Fig. 3. Simple topology.

A. Accuracy and Completeness

In [4] we cast the problem of detecting compromised routers as a failure detector with *accuracy* and *completeness* properties. There are two steps in showing the accuracy and completeness of χ:

- Showing that TV is correct.
- Showing that χ is accurate and complete assuming that TV is correct.

As we assume that adjacent routers cannot be compromised in our threat model, we show in Appendix II and Appendix III that if TV is correct, then χ is *2–accurate* and *2–complete*, where 2 indicates the length of detection: A link consists of two routers is detected as a result. This assumption eliminates consorting faulty routers that collude together to produce fraudulent traffic information to hide their fault. This assumption can be relaxed to the case of $k > 1$ adjacent faulty routers by monitoring every output interfaces of the neighbors k hops away and disseminating the traffic information to all neighbors within diameter k hops. This is the same approach that we used in [4], and it increases the overhead of detection.

We discuss traffic validation next.

B. Traffic Validation Correctness

Any failure of detecting malicious attack by TV results in a false negative, and any misdetection of legitimate behavior by TV results in a false positive.

Within the given system model of Section IV, the example TV predicate in Section V-A is correct. However, the system model is still simplistic. In a real router, packets may be legitimately dropped due to reasons other than congestion: for example, errors in hardware, software or memory, and transient link errors. Classifying these as arising from a router being compromised might be a problem, especially if they are infrequent enough that they would be best ignored rather than warranting repairing the router or link.

A larger concern is the simple way that a router is modeled in how it internally multiplexes packets. This model is used to compute time stamps. If the time stamps are incorrect, then TV could decide incorrectly. We hypothesize that a sufficiently accurate timing model of a router is attainable, but have yet to show this to be the case.

A third concern is with clock synchronization. This version of TV requires that all the routers feeding a queue have synchronized clocks. This requirement is needed to ensure that the packets are interleaved correctly by the model of the router.

The synchronization requirement is not necessarily daunting; the tight synchronization is only required by routers adjacent to the same router. With low level timestamping of packets, and repeated exchanges of time [27], it should be straightforward to synchronize the clocks sufficiently tightly.

Other representations of collected traffic information and TV that we have considered have their own problems of false positives and false negatives. It is an open question as to the best way to represent TV. We suspect any representation will admit some false positives or false negatives.

C. Overhead

1) Computing Fingerprints: The main overhead of protocol χ is in computing a fingerprint for each packet. This computation must be done at wire speed. Such a speed has been demonstrated to be attainable.

In our prototype, we implemented fingerprinting using UHASH [28]. [29] demonstrated UHASH performance of over 1Gbps on a 700Mhz Pentium III processor when computing a 4 byte hash value. This performance could be increased further with hardware support.

Network processors are designed to perform highly parallel actions on data packets [30]. For example, Feghali *et al.* [31] presented an implementation of well known private-key encryption algorithms on the Intel IXP28xx network processors to keep pace with a 10Gigabit/sec forwarding rate. Furthermore, Sanchez *et al.* [32] demonstrated hardware support to compute fingerprints at wire speed of high speed routers (OC-48 and faster).

2) State Requirement: Let N be the number of routers in the network, and R be the maximum number of links incident on a router. Protocol χ requires a router to monitor the path segments that are at most two hops away. By construction, this is $O(R^2)$. State is kept for each of these segments. The TV predicate in Section V-A requires a time stamp and the packet size be kept for each packet that traversed the path segment. As a point of comparison, `WATCHERS` [3] requires $O(RN)$ state, where each individual router keeps seven counters for each of its neighbors for each destination.

3) Computing TV: The time complexity to compute TV depends on the size of the traffic information collected and received from the neighbors that are within 2 hops, and so it depends on the topology and the traffic volume on the network. If traffic information stores the packet fingerprints in order of increasing timestamps, then a straightforward implementation of traffic validation exists.

In our prototype, which is not optimized, *TV* computation had an overhead of between 15 to 20 milliseconds per validation round.

4) Control Message Overhead: Protocol χ collects traffic information and exchanges them periodically using the monitored network infrastructure. Lets say each fingerprint and timestamp are both 4 bytes. Then, message overhead is 8 bytes per packet. If we assume that the average packet size is 800 bytes, then the bandwidth overhead of protocol χ is 1%.

5) Clock Synchronization: Similar to all previous detection protocols, χ requires synchronization to agree on a time interval to collect traffic information. For a router r, all neighboring routers of r need to synchronize with each other to agree on when and for how long the next measurement interval τ will be.

Clock synchronization overhead is fairly low. For example, external clock synchronization protocol NTP [33] can provide accuracy within 200 microseconds in local area networks. It requires two messages of 90 bytes per transaction and the rate of transactions can be once per minute to once per 17 minutes. [34] presented an internal clock synchronization protocol (RTNP) that maintains an accuracy within 30 microseconds by updating the clocks once every second.

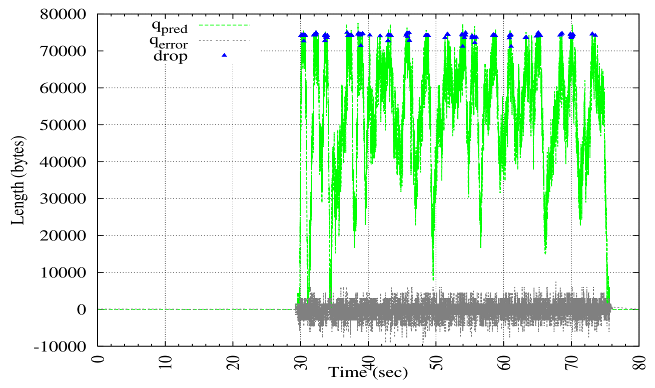
6) Key Distribution: To protect against *p-faulty* routers tampering the messages containing traffic information, χ requires digital signatures or message authentication codes. Thus, there is an issue of *key distribution* depending on the cryptographic tools that are used.

VII. Experiences

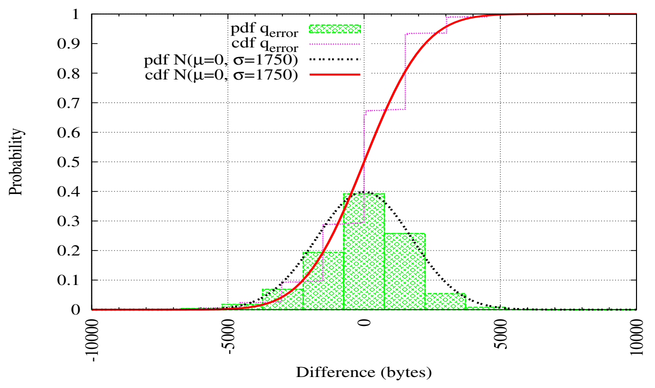
We have implemented and experimented with protocol χ in the Emulab [35], [36] testbed. In our experiments, we used the simple topology shown in Figure 3. The routers were Dell PowerEdge 2850 PC nodes with a single 3.0 GHz 64-bit Xeon processor, 2GB of RAM, and they were running Redhat-Linux-9.0 OS software. Each router except for r_1 was connected to three LANs to which user machines were connected. The links between routers were configured with 3 Mbps bandwidth, 20 msec delay, and 75000 byte capacity FIFO queue.

Each pair of routers share secret keys, furthermore integrity and authenticity against the message tampering is provided by message authentication codes.

The validation time interval τ was set to 1 second and the upper bound on the time to forward traffic information Δ was set to 300 milliseconds. At the end of each second, the routers exchanged traffic information corresponding the last validation interval , and evaluated the TV predicate after $2\Delta = 600$ milliseconds. Each run in an experiment consisted of an execution of 80 seconds. During the first 30 seconds, we generated no traffic to allow the routing fabric to initialize. Then, we generated 45 seconds of traffic.



(a) Queue Length.



(b) Distribution of q_{error}.

Fig. 4.

A. Experiment 1: Protocol χ with no attack

We first investigated how accurately the protocol predicts the queue lengths of the monitored output interfaces. We considered the results for the output interface Q of r_1 associated with the link $\langle r_1, r_2\rangle$. Background traffic was created to make $\langle r_1, r_2\rangle$ a bottleneck. 20% of the bottleneck bandwidth was consumed by constant bit rate traffic, another 20% by short lived http traffic, and the rest by long lived ftp traffic.

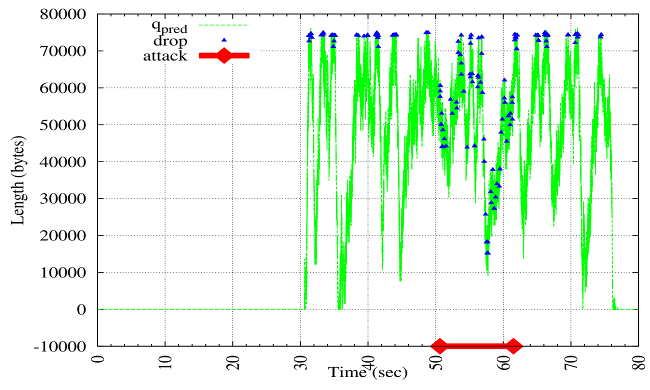
The result of one run is shown in Figure 4(a). q_{pred} is the predicted queue length of Q computed by router r_2 executing the protocol χ. q_{act}, which is the actual queue length of Q recorded by router r_1, is not shown in the graph because it is so close to q_{pred}. Instead, the difference $q_{error} = q_{act} - q_{pred}$, is plotted; its value ranges approximately from -7500 bytes to 7500 bytes. Packet drops—all due to congestion—are marked with triangles.

Next, we examine the distribution of q_{error}. In Figure 4(b), the probability distribution and cumulative distribution functions of q_{error} are plotted. It is clustered around the multiples of 1500 bytes, since it is the maximum transmission unit and most frequent packet size of the traffic. Computing the mean, μ, and the standard deviation, σ, of this data, the corresponding normal distribution functions are also shown in the graph. It turns out that the distribution of q_{error} can be approximated by a normal distribution $N(\mu, \sigma)$.

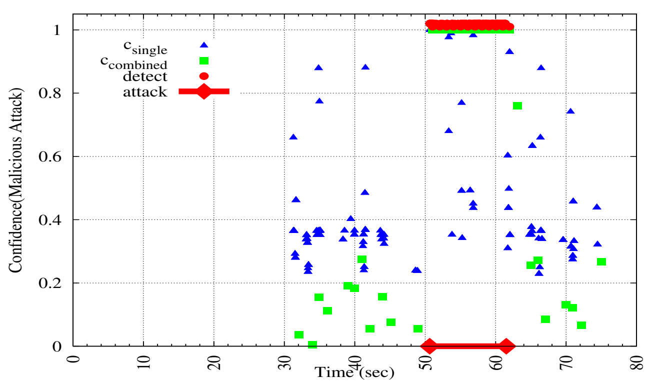
We expected many different causes contribute to q_{error}: inaccurate clock synchronization, scheduling delays, internal processing delays, and so on. It turns out that scheduling and clock synchronization inaccuracy are the dominant factors. In terms of scheduling, all routers are running Linux with a programmable interval timer of 1024 Hz . This results in a scheduling quantum of roughly 1 millisecond. We verified the affect of the scheduling quantum by changing the frequency to 100Hz frequency, and we observed that the variance of the distribution of q_{error} changed accordingly. For clock synchronization, we used NTP [33] to synchronize the routers' clocks, but it takes a long time for the NTP daemon to synchronize the routers' clocks to within a few milliseconds. So, we used a different strategy: once a second we reset each router's clock to the NTP server's clock. This resulted in the clocks being synchronized to within 0.5 msec. Finally, the processing delay of the packets within a router is typically less than 50 microseconds. So, it does not introduce significant uncertainty compared to those factors.

B. Experiment 2: False positives

In the second experiment, we first ran a training run to measure the mean and standard deviation of q_{error}. We found $\mu = 0$ and $\sigma = 1750$. We then ran protocol χ under high traffic load for more than one hour, which generated more than half a million packets. Approximately 4,000 validation rounds occurred in this run, and approximately 16,000 packets were dropped due to congestion. Choosing significance levels $s^{level}_{single} = 0.999$ and $s^{level}_{combined} = 0.9$, there were eight false positives generated by the single packet drop test and two false positives generated by the combined packet drop test. Both are lower than one would expect given the number of samples. We suspect that the lower false positive rate for the single packet drop test is because the distribution of q_{error} is not truly a normal distribution, and the lower false positive rate for the combined packet drop test is because the test is not done on a simple random sample. We are investigating this further. In all of the subsequent experiments, we used the same mean, standard deviation, and two significance levels given here.

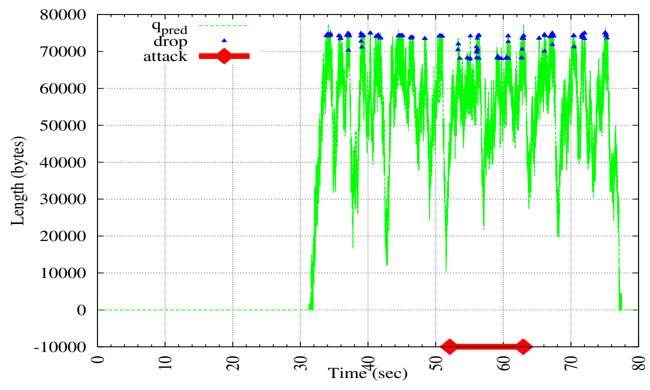


(a) Queue length.

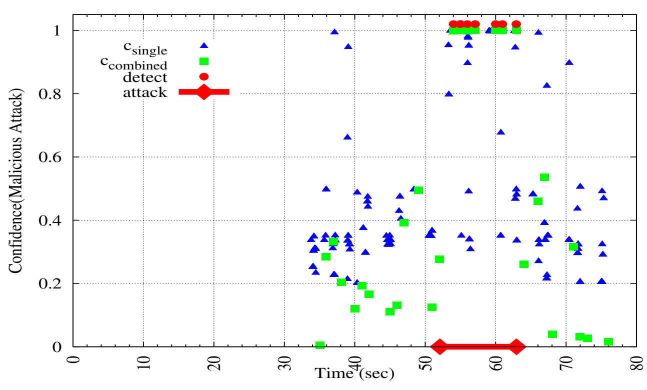


(b) Statistical test results.

Fig. 5. Attack 1: *Drop 20% of the selected flows.*



(a) Queue length.



(b) Statistical test results.

Fig. 6. Attack 2: *Drop the selected flows when queue is 90% full.*

C. Experiment 3: Detecting attacks

We then experimented with the ability of protocol χ to detect attacks. In these experiments, the router r_1 is compromised to attack the traffic selectively in various ways, targeting the chosen two *ftp* flows. The duration of the attack is indicated with line bounded by diamonds in the figures, and a detection is indicated by a filled circle.

For the first attack, the router r_1 was instructed to drop 20% of the selected flows for 10 seconds. Predicted queue length and the confidence values for each packet drop can be seen in Figure 5(a) and Figure 5(b). As shown in the graph, during the attack, protocol χ detects the failure successfully.

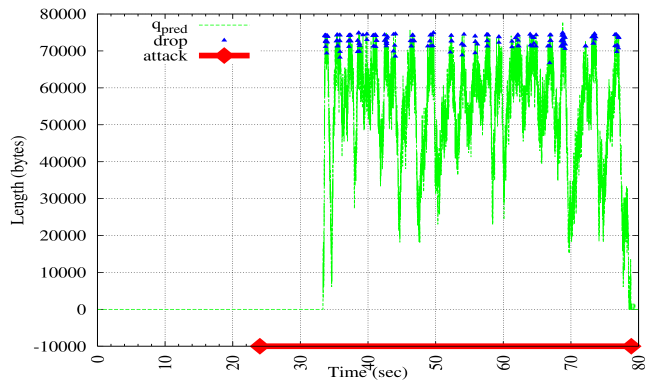
In the second attack, router r_1 was instructed to drop packets in the selected flows when the queue was at least 90% full. Protocol χ was able to detect the attack and raised alarms, as shown in Figure 6.

Next, we increase the threshold that r_1 attacks to 95%. No single drop test has enough confidence to raise an alarm because all the drops are very close to the q_{limit}. However, χ raised alarms with the combined drops test. Even few additional packets were dropped, the impact on the TCP flows of this attack was significant. Both attacked flows' bandwidth usage dropped more than 35%, and their share was used by the other flows.

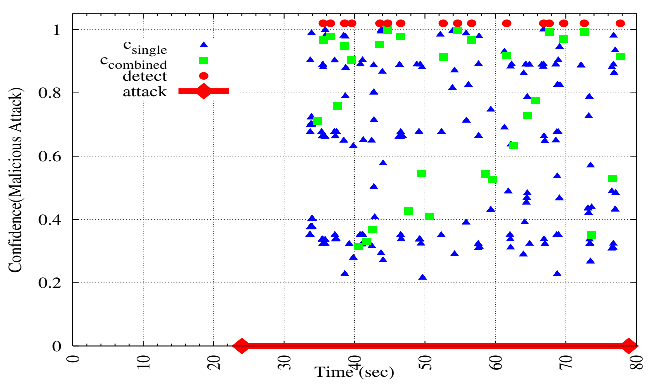
Last of all, we looked in the SYN attack preventing a selected host establishing connection with any server: The router r_1 was instructed to drop all SYN packets from a targeted host, which tries to connect to an ftp server. In Figure 8, five SYN packets, which are marked with circles, are maliciously dropped by r_1. Except for the second SYN packet drop, all malicious drops raised an alarm. The second SYN is dropped when the queue is almost full, and so the confidence value is not significant to differentiate it from the other packet drops due to congestion.

D. Protocol χ vs. Static threshold

We argued earlier the difficulties of using static thresholds of dropped packets for detecting malicious intent. We illustrate this difficulty with the run shown in Figure 6. Recall that in this run, the router dropped packets

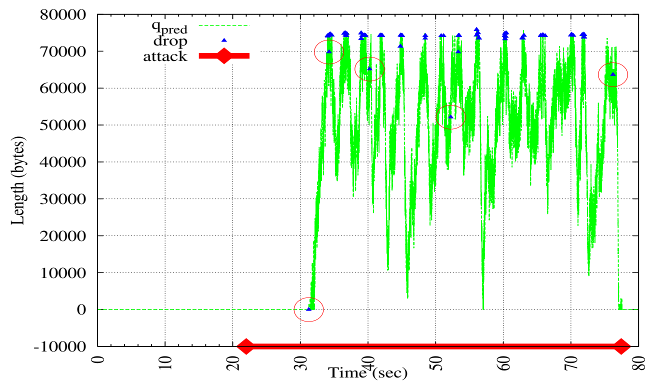


(a) Queue length.

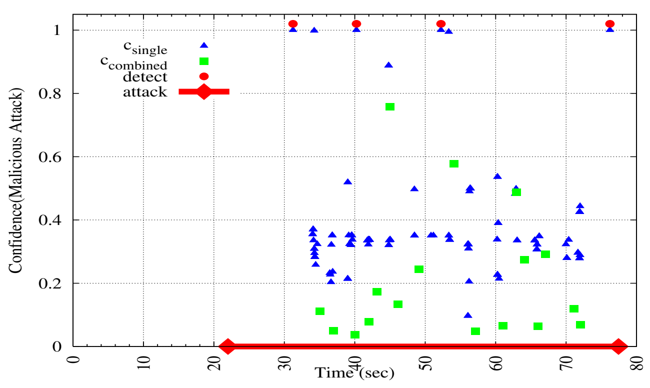


(b) Statistical test results.

Fig. 7. Attack 3: *Drop the selected flows when queue is 95% full.*



(a) Queue length.



(b) Statistical test results.

Fig. 8. Attack 4: *Target a host trying to open a connection by dropping SYN packets.*

only when the output queue was at least 90% full. Before time 52, the router behaved correctly, and 2.1% of the packets were dropped due to congestion. During the time period from 52 to 64, the router maliciously dropped packets, but only 1.7% of the packets were dropped (some due to congestion and some due to the attack). This may seem counterintuitive: fewer packets were dropped due to congestion during the period that the queues contained more packets. Such a nonintuitive behavior doesn't happen in every run, but the dynamics of the network transport protocol led to this behavior in the case of this run. So, for this run, there is no static threshold that can be used to detect the period during which the router was malicious. A similar situation occurs in the highly-focused SNY attack of Figure 8.

In contrast, protocol χ can detect the malicious behaviors because it measures the router's queues, which are determined by the dynamics of the network transport protocol. Protocol χ can have false positive and false negative detections, but the probability of such false detections can be controlled by setting a significance level for the statistical tests upon which χ is built. A static threshold can not be used in the same way.

VIII. Non-deterministic Queuing

As described, our traffic validation technique assumes a deterministic queuing discipline on each router; *first in first out* (FIFO) with tail-drop. While this is a common model, in practice, real router implementations can be considerably more complex – involving switch arbitration, multiple layers of buffering, multicast scheduling, etc. Of these, the most significant for our purposes is the non-determinism introduced by active queue management (AQM), such as *random early detection* (RED) [37], *proportional integrator* (PI) [38], and *random exponential marking* (REM) [39]. In this section, we describe how protocol χ can be extended to validate traffic in AQM environments. We focus particularly on RED, since this is the most widely-known and widely-used of such mechanisms.[6]

[6]Although RED is universally *implemented* in modern routers, it is still unclear how widely it is actually used.

RED was first proposed by Floyd and Jacobson in the early 1990s to provide better feedback for end-to-end congestion control mechanisms. Using RED, when a router's queue becomes full enough that congestion may be imminent, a packet is selected at random to signal this condition back to the sending host. This signal can take the form of a bit marked in the packet's header and then echoed back to the sender – *Explicit Congestion Notification* (ECN) [40], [41] – or can be indicated by dropping the packet.[7] If ECN is used to signal congestion, then protocol χ, as presented in Section V, works perfectly. If not, then RED will introduce non-deterministic packet losses that may be misinterpreted as malicious activity.

In the remainder of this section, we explain how RED's packet selection algorithm works, how it may be accommodated into our traffic validation framework, and how well we can detect even small attacks in a RED environment.

A. Random Early Detection

RED monitors the average queue size, q_{avg}, based on an exponential weighted moving average:

$$q_{avg} := (1-w)q_{avg} + w \cdot q_{act} \tag{1}$$

where q_{act} is the actual queue size, and w is the weight for a low-pass-filter.

RED uses three more parameters: q^{th}_{min}, minimum threshold; q^{th}_{max}, maximum threshold; and p_{max}, maximum probability. Using q_{avg}, RED dynamically computes a dropping probability in two steps for each packet it receives. First, it computes a interim probability, p_t:

$$p_t = \begin{cases} 0 & \text{if } q_{avg} < q^{th}_{min} \\ p_{max}\frac{q_{avg}-q^{th}_{min}}{q^{th}_{max}-q^{th}_{min}} & \text{if } q^{th}_{min} < q_{avg} < q^{th}_{max} \\ 1 & \text{if } q^{th}_{max} < q_{avg} \end{cases}$$

Further, the RED algorithm tracks the number of packets, *cnt*, since the last dropped packet. The final dropping probability, p, is specified to increase slowly as *cnt* increases:

$$p = \frac{p_t}{1 - \mathit{cnt} \cdot p_t} \tag{2}$$

Finally, instead of generating a new random number for every packet when $q^{th}_{min} < q_{avg} < q^{th}_{max}$, a suggested optimization is to only generate random numbers when a packet is dropped [37]. Thus, after each RED-induced packet drop, a new random sample, *rn*, is taken from a uniform random variable $R = \mathit{Random}[0,1]$. The first packet whose p value is larger than *rn* is then dropped, and a new random sample is taken.

B. Traffic Validation for RED

Much as in Section V-A our approach is to predict queue sizes based on summaries of their inputs from neighboring routers. Additionally, we track how the predicted queue size impacts the likelihood of a RED-induced drop and use this to drive two additional tests: one for the uniformity of the randomness in dropping packets and one for the distribution of packet drops among the flows.[8] In effect, the first test is an evaluation of whether the *distribution* of packet losses can be explained by RED and tail-drop congestion alone, while the second evaluates if the particular *pattern* of losses (their assignment to individual flows) is consistent with expectation for traffic load.

[7]ECN-based marking is well-known to be a superior signaling mechanism [42], [43]. However, while ECN is supported by many routers (Cisco and Juniper) and end-systems (Windows Vista, Linux, Solaris, NetBSD, etc) it is generally not enabled by default and thus it is not widely deployed in today's Internet [44], [45].

[8]Consistent with our assumption that the network is under a single administrative domain (Section IV, we assume that all RED parameters are known).

Packet		fp_1	fp_2	fp_3	fp_4	fp_5	fp_6	fp_7	fp_8	fp_9	.	.	.	fp_n
Drop probability		p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	.	.	.	p_n
Outcome		*TX*	*TX*	*DR*	*TX*	*TX*	*TX*	*TX*	*DR*	*TX*	.	.	.	*TX*
Random number	rn_1			rn_2					rn_3					

Fig. 9. A set n packets. Each packet fp_i is associated with a drop probability p_i and the outcome is either transmitted(TX) or dropped(DR) based on the random number generated during the last packet drop.

1) Testing the uniformity packet drops: In Figure 1, router r_d monitors the queue size of router r and detects whether each packet is dropped or transmitted. Given the RED algorithm and the parameters, r_d now can estimate q_{avg}, the average queue size in Formula 1; *cnt*, the count since the last dropped packet; and finally p, the dropping probability in Formula 2 for each packet as in Figure 9. All of these computations are deterministic based on observed inputs.

The router r drops a packet fp_i if its p_i value exceeds the random number rn_x that it generated at the most recent packet drop. So, r_d expects that rn_x is between p_{i-1} and p_i. For example in Figure 9:

- fp_3 is dropped: $p_2 < rn_1 < p_3$.
- fp_8 is dropped: $p_7 < rn_2 < p_8$.

Since each packet drop should be a sample of a uniform random distribution, we can detect deviations from this process via statistical hypothesis testing. In particular, we use the Chi-Square test to evaluate the hypothesis that the observed packet losses are a good match for a uniform distribution [26]. Once the *Chi-Square* value[9] is computed, then the corresponding critical value can be used as the confidence value $c_{randomness}$ to reject the hypothesis, which means the outcome is a result of non-uniform distribution and/or a detection of malicious activity. Thus, a malicious router is detected if the confidence value $c_{randomness}$ is at least a target significance level $s^{level}_{randomness}$.

2) Testing the distribution of packet drops among flows: One of the premises of RED [37] is that the probability of dropping a packet from a particular connection is proportional to that connection's bandwidth usage. We exploit this observation to evaluate if the particular pattern of packet losses – even if not suspicious in their overall number – is anamolous with respect to per-flow traffic load.

This test requires per flow state for counting the number of received packets and dropped packets per flow during $q^{th}_{min} < q_{avg} < q^{th}_{max}$. Once again, we use the Chi-Square test to evaluate the distribution of packet losses to flows[10] Once the *Chi-Square* value is computed, then the corresponding critical value can be used as the confidence value $c_{drop/flow}$ to reject the hypothesis, which means that the distribution of packet drops among the flows is not as expected. A malicious router is detected if the confidence value $c_{drop/flow}$ is at least a target significance level $s^{level}_{drop/flow}$.

C. Experiences

We have experimented with protocol χ with this new traffic validation in a RED environment using the same setup presented in Section VII. The capacity of the queue, q^{th}_{limit}, is 75000 bytes. In addition, the RED parameters, as in Section VIII-A, are configured as following: the weight for the low-pass-filter, $w = 0.5$; the minimum threshold, $q^{th}_{min} = 30,000$ bytes; the maximum threshold, $q^{th}_{max} = 60,000$ bytes; the maximum probability, $p_{max} = 0.02$.[11]

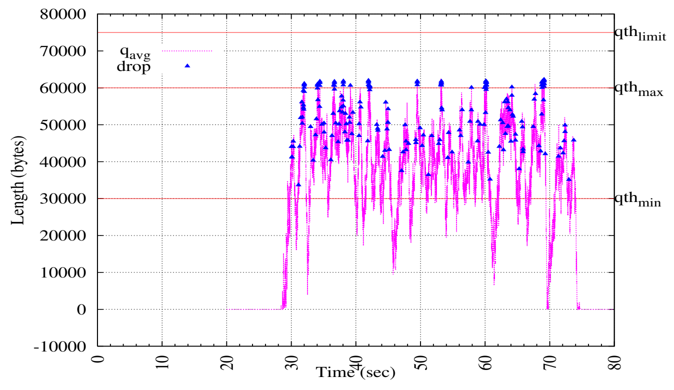
For the *packet drop uniformity test*, a window of 30 packet drops is used. The *distribution of packet drops to flows test* examines a window of 15 seconds. Experimentally we find that smaller windows lead to false positives, but larger windows do not improve the results notably. A more sophisticated version of our algorithm could adapt the window size in response to load to ensure a given level of confidence.

1) Experiment 1: False positives: The result of one run is shown in Figure 10(a). q_{avg} is the predicted average queue length of Q computed by router r_2. Packet losses are also marked with triangles. The corresponding confidence values can be seen in Figure 10(b).

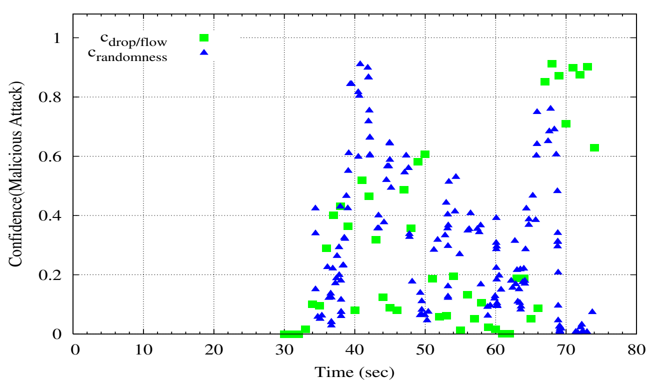
[9]*Chi-Square* $= \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$, where O_i is the observed frequency of bin i; E_i is the expected frequency of bin i; and k is the number of bins.

[10]Short lived flows with couple of tens of packets are ignored unless the drop rate is 100%. Otherwise, a few packet drops from a short lived flow lead to false detection.

[11]Setting the parameters is inexact engineering. We used the guidelines presented in [37] and/or used our intuition selecting these values.

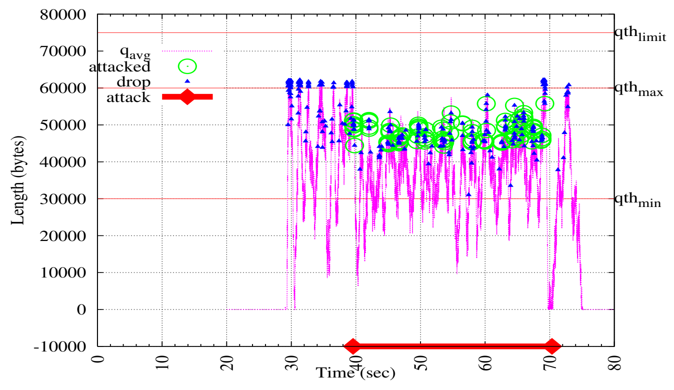


(a) Average queue length.

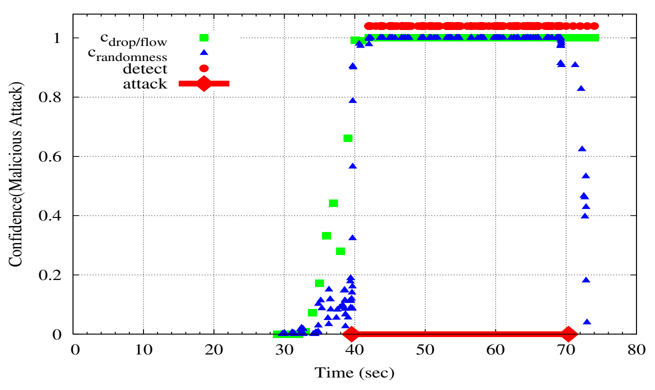


(b) Statistical test results.

Fig. 10. Without attack.



(a) Average queue length.



(b) Statistical test results.

Fig. 11. Attack 1: *Drop the selected flows when the average queue size is above* $45,000$ *bytes.*

We executed protocol χ under high traffic load for more than half an hour. Choosing significance levels aggressively $s^{level}_{randomness} = 0.999$ and $s^{level}_{drop/flow} = 0.999$, we have not observed any false positives.

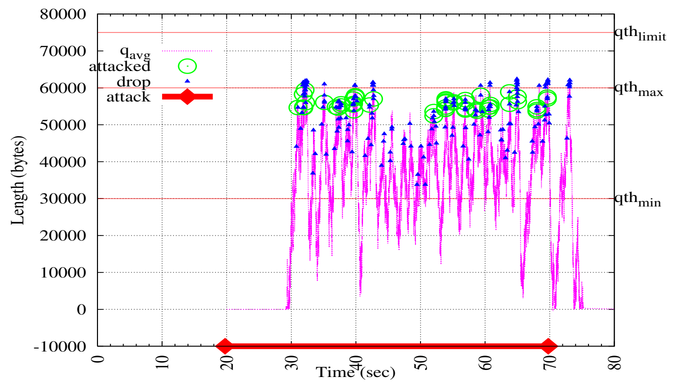
2) Experiment 2: Detecting attacks: Next we examined how effectively protocol χ detects various attacks. In these experiments, the router r_1 is compromised to attack the traffic selectively in various ways, targeting *ftp* flows from a chosen subnet. The duration of the attack is indicated with line bounded by diamonds in the figures, and a detection is indicated by a filled circle.

For the first attack, the router r_1 drops the packets of the selected flows for 30 seconds when the average queue size computed by RED is above $45,000$ bytes. The predicted average queue size and the confidence values can be seen in Figure 11. As shown in the graph, during the attack, protocol χ detects the failure successfully.

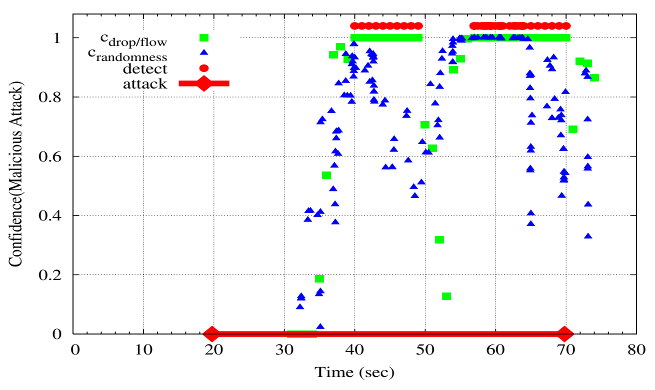
As queue occupancy grows, the RED algorithm drop packets with higher probability and thus provides more "cover" for attackers to drop packets without being detected. We explore this property in the second attack, in which router r_1 was instructed to drop packets in the selected flows when the average queue was at least $54,000$ bytes, which is very close to the maximum threshold, $q^{th}_{max} = 60,000$ bytes. As shown in Figure 12, protocol χ was still able to detect the attack and raised alarms, except between 50 and 56 seconds. The reason is that between 44 and 50 seconds the compromised router did not drop any packets maliciously.

In the third and fourth attacks, we explore a scenario in which a router r_1 only drops a small percentage of the packets in the selected flows. For example, in the third attack 10% percent of packets are dropped (see Figure 13) and 5% in the fourth attack (see Figure 14). Even though relatively few packets are dropped, the impact on TCP performance is quite high, reducing bandwidth between 40% and 30%. Since only a few packets are maliciously dropped, the *packet drop uniformity test* does not detect any anomaly. However, since these losses are focused on a small number of flows, they are quickly detected using the second test.

Finally, we explained a highly-selective attack in which the router r_1 was instructed to only drop TCP SYN

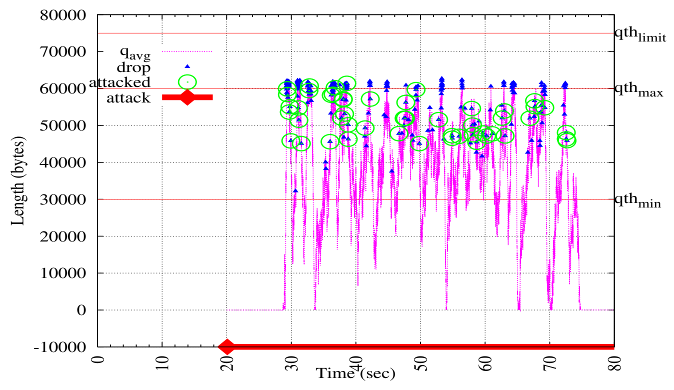


(a) Average queue length.

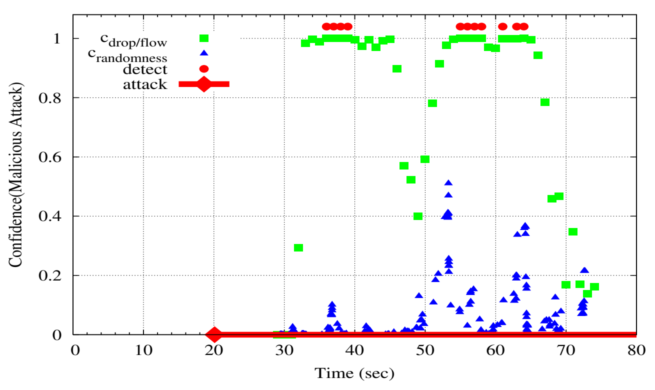


(b) Statistical test results.

Fig. 12. Attack 2: *Drop the selected flows when the average queue size is above* 54,000 *bytes.*



(a) Average queue length.



(b) Statistical test results.

Fig. 13. Attack 3: *Drop 10% of the selected flows when the average queue size is above* 45,000 *bytes.*

packets from a targeted host, which tries to connect to an ftp server. In Figure 15, four SYN packets, which are marked with circles, are maliciously dropped by r_1. Since *all* the observed packets of the attacked flow are dropped, which is statistically unexpected given the RED algorithm, protocol χ still raises an alarm.

IX. Conclusion

To the best of our knowledge, this paper is the first serious attempt to to distinguish between a router dropping packets maliciously and a router dropping packets due to congestion. Previous work has approached this issue using a static user-defined threshold, which is fundamentally limiting.

Using the same framework as our earlier work (which is based on a static user-defined threshold) [4], we developed a compromised router detection protocol χ that dynamically infers, based on measured traffic rates and buffer sizes, the number of congestive packet losses that will occur. Once the ambiguity from congestion is removed, subsequent packet losses can be attributed to malicious actions. Because of nondeterminism introduced by imperfectly synchronized clocks and scheduling delays, protocol χ uses user-defined significance levels, but these values are independent of the properties of the traffic. Hence, this approach does not suffer from the limitations of static thresholds.

We evaluated the effectiveness of Protocol χ through an implementation and deployment in a small network. We show that even fine grain attacks, such as stopping a host from opening a connection by discarding the SYN packet, are detected.

Appendix I
Specification

Similar to the specification that we have defined in [4], we cast the problem as a failure detector with *accuracy* and *completeness* properties.

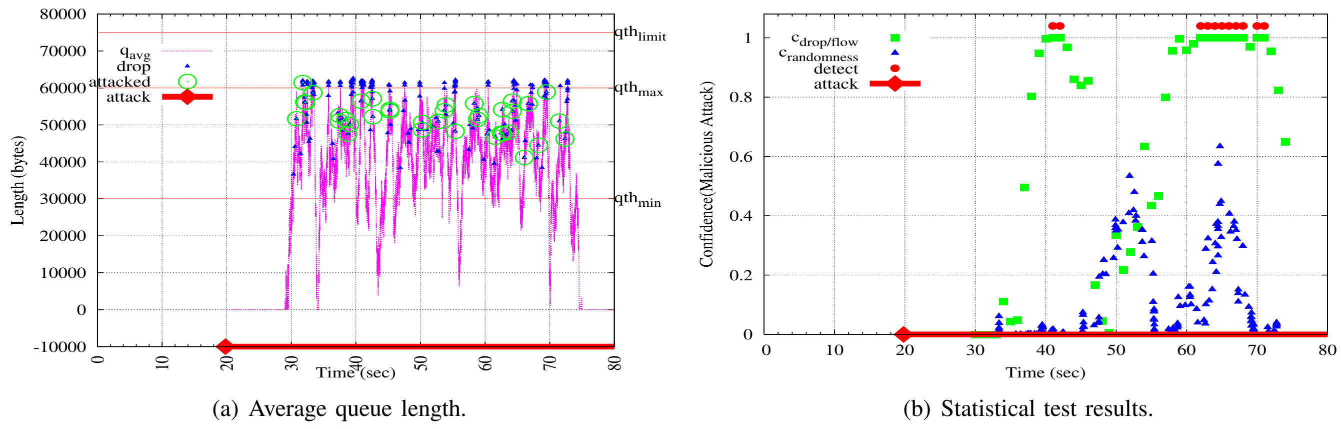


Fig. 14. Attack 4: *Drop 5% of the selected flows when the average queue size is above* $45,000$ *bytes.*

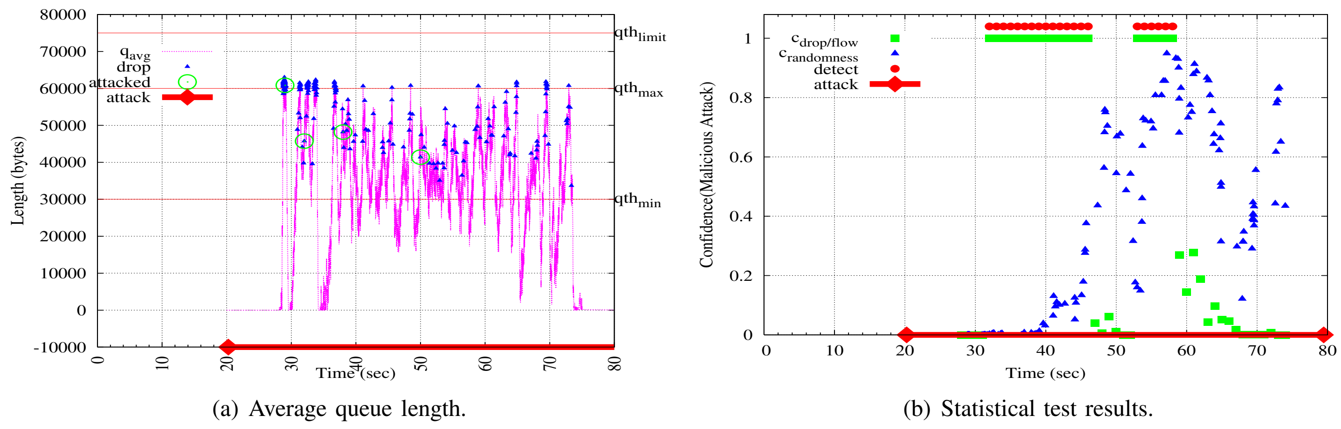


Fig. 15. Attack 5: *Target a host trying to open a connection by dropping SYN packets.*

- a–Accuracy: A failure detector is *a–Accurate* if, whenever a correct router suspects (π, τ), then $|\pi| \leq a$ and some router $r \in \pi$ was faulty in π during τ.

We use the term *t-faulty* (that is, *traffic* faulty) to indicate a router that drop packets from transit traffic and the term *p-faulty* (that is, *protocol* faulty) to indicate a router that behaves arbitrarily with respect to the detection protocol. A *faulty* router is one that is t-faulty, p-faulty or both. As before, we will add the phrase "in π" to indicate that the faulty behavior is with respect to traffic that transits the path π. The a–Accuracy requirement can result in a detection if a router is either p-faulty or t-faulty.

- a–Completeness: A failure detector is *a–Complete* if, whenever a router r is t-faulty at some time t, then all correct routers eventually suspect (π, τ) for some path segment $\pi : |\pi| \leq a$ such that r was t-faulty in π at t, and for some interval τ containing t.

Distinguishing between p-faulty and t-faulty behavior is useful because, while it is important to detect routers that are t-faulty, it isn't as critical to detect routers that are only p-faulty: routers that are only p-faulty are not dropping packets maliciously.

Given an accurate and complete TV predicate, protocol χ is *2–accurate* and *2–complete*.

Appendix II
Accuracy

When a correct router e'' receives a suspicious link $\ell = \langle e, e' \rangle$ announcement originated by router e, e'' detects ℓ as faulty. Then there must be at least one faulty router in ℓ:

- If e is *faulty*, and it announces its link ℓ as faulty; indeed ℓ has a faulty router: $e \in \ell$. A *p-faulty* router can always announce its link ℓ as faulty.

- If e is correct, and suspects its neighbor e' announcing its link ℓ as faulty, then e' must be faulty. We show this by considering each detection in Section. V-B.
  - D-Ia: Assume e' is correct, and it sends its traffic information *Tinfo*$(r', Q_{in}, \langle e', e, r_d \rangle, \tau)$ to router e at the end of validation time interval τ. The message must be delivered to e in Δ time, which is a contradiction of e being correct yet not receiving this message.
  - D-Ib: Assume e' is correct, and sends digitally signed traffic information which is consistent and valid. Correct router e validates the signature and the consistency of the traffic information. This contradicts e suspects e'.
  - D-IIa: Assume e' is correct. Then one of the following is true: 1) e' received a traffic information from r_{s_*} in Δ time, verified it and forwarded it to e in the next Δ time. This contradicts with correct router e not being received the message. 2) e' did not verified the traffic information from r_{s_*} or had not received the message in Δ time. Then it should have detect $\ell = \langle r_{s_*}, e' \rangle$ and announced the detection. This contradicts with correct router e not receiving the detection announcement.
  - D-IIb: Assume e' is correct, and forwards traffic information to e only if it validates the signature. Then the correct router e validates the signature. This contradicts the failure of the digital signature verification.
  - D-IIc: Assume e' is correct, and forwards traffic correctly. Since e' is correct, all traffic information of S, which e' sent to e, is verified by e'. With the input of S verified by correct router e' and the input of D collected by e, TV predicate evaluates to *true*. This contradicts with TV evaluating to *false*.

All detections by protocol χ are 2-path segments. Hence, it is *2-accurate*.

Appendix III
Completeness

If a router e is t-faulty[12] at some time t, then all correct routers eventually suspect $\langle \ell, \tau \rangle$ for some link ℓ such that $e \in \ell$ and e was t-faulty at t, and for some interval τ containing t.

Let e have dropped packets maliciously from the traffic passing through itself forwarding to e' during τ containing t. At the end of traffic validation round τ, e' will validate Q associated with ℓ.

As we assume that adjacent routers cannot be compromised in our threat model, all neighbors of e are correct and collect traffic information appropriately during τ. At the end of τ, they send these information to e to forward to e'.

Then one of the following is true:

- D-IIc: e passes these information to e'. The complete TV evaluates to *false* with these correct inputs. So e' detects $\ell = \langle e, e' \rangle$, where $e \in \ell$.
- D-IIb: e passes these information to e' after tampering the content hiding the attack. e' fails to verify the signatures, so e' detects $\ell = \langle e, e' \rangle$, where $e \in \ell$.
- D-IIa: e passes its own copy of traffic information to e' hiding the attack. Then e' expects e to detect r_{s_*} whose traffic information has not been forwarded to e'. 1) If e detects $\ell = \langle r_{s_*}, e \rangle$, then $e \in \ell$. 1) If e fails to detect $\ell = \langle r_{s_*}, e \rangle$, then e' detects $\ell = \langle e, e' \rangle$, where $e \in \ell$.
- D-IIa: e does not send any traffic information in S to e'. Due to the timeout mechanism, after 2Δ time, e' detects $\ell = \langle e, e' \rangle$, where $e \in \ell$.

Acknowledgment

The first author would like to thank Mustafa Arisoylu for the insightful discussions about the dynamics of RED algorithm.

[12]As in [4], the protocol we develop assumes that the terminal routers are correct. This assumption is common to all detection protocols. The argument is based on *fate sharing*. If a terminal router is compromised then there is no way to determine what traffic was injected or delivered in the system. Thus, a compromised terminal router can always invisibly disrupt traffic sourced or sinked to its network. One could place terminal routers on each workstation thus limiting the damage they can wreak to only a workstation.

References

[1] X. Ao, "Report on DIMACS Workshop on Large-Scale Internet Attacks," http://dimacs.rutgers.edu/Workshops/Attacks/internet-attack-9-03.pdf, Sep. 2003.

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

[3] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson, "Detecting disruptive routers: A distributed network monitoring approach," in *Proc. of the IEEE Symposium on Security and Privacy*, May 1998, pp. 115–124.

[4] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage, "Detecting and isolating malicious routers," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 3, pp. 230–244, Jul-Sep 2006.

[5] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. Katz, "Listen and Whisper: Security Mechanisms for BGP," in *Proc. of NSDI*, Mar. 2004.

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

[7] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: A secure on-demand routing protocol for ad hoc networks," in *The 8th ACM Int. Conf. on MobiCom*, Sep 2002.

[8] B. R. Smith and J. Garcia-Luna-Aceves, "Securing the border gateway routing protocol," in *Proc. Global Internet'96*, Nov 1996.

[9] S. Cheung, "An efficient message authentication scheme for link state routing," in *ACSAC*, 1997, pp. 90–98.

[10] M. T. Goodrich, "Efficient and secure network routing algorithms," Jan 2001, provisional patent filing.

[11] R. Perlman, "Network layer protocols with byzantine robustness," Ph.D. dissertation, MIT LCS TR-429, Oct. 1988.

[12] V. N. Padmanabhan and D. Simon, "Secure traceroute to detect faulty or malicious routing," *SIGCOMM Comp. Comm. Review*, vol. 33, no. 1, pp. 77–82, 2003.

[13] I. Avramopoulos and J. Rexford, "Stealth probing: Efficient data-plane security for ip routing," in *Proc. USENIX Annual Technical Conference*, June 2006.

[14] S. Cheung and K. N. Levitt, "Protecting routing infrastructures from denial of service using cooperative intrusion detection," in *NSPW '97: Proc. of the workshop on New security paradigms*, 1997, pp. 94–106.

[15] J. R. Hughes, T. Aura, and M. Bishop, "Using conservation of flow as a security mechanism in network protocols," in *IEEE Symp. on Security and Privacy*, 2000, pp. 132–131.

[16] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage, "Fatih: Detecting and isolating malicious routers," in *Proc. of the 2005 Int. Conf. on Dependable Systems and Networks (DSN'05)*, 2005, pp. 538–547.

[17] A. Kuzmanovic and E. W. Knightly, "Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants," in *Proc. of the SIGCOMM '03*, 2003, pp. 75–86.

[18] M. Mathis, J. Semke, and J. Mahdavi, "The macroscopic behavior of the TCP congestion avoidance algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 3, pp. 67–82, 1997.

[19] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: a simple model and its empirical validation," in *Proc. of the SIGCOMM '98*, 1998, pp. 303–314.

[20] M. Yajnik, S. B. Moon, J. F. Kurose, and D. F. Towsley, "Measurement and modeling of the temporal dependence in packet loss." in *Proc. of the INFOCOM '99*, 1999, pp. 345–352.

[21] N. Cardwell, S. Savage, and T. E. Anderson, "Modeling TCP latency." in *Proc. of the INFOCOM '00*, 2000, pp. 1742–1751.

[22] E. Altman, K. Avrachenkov, and C. Barakat, "A stochastic model of TCP/IP with stationary random losses," in *Proc. of the SIGCOMM '00*, 2000, pp. 231–242.

[23] W. Jiang and H. Schulzrinne, "Modeling of packet loss and delay and their effect on real-time multimedia service quality," in *Proc. NOSSDAV*, 2000.

[24] T. J. Hacker, B. D. Noble, and B. D. Athey, "The effects of systemic packet loss on aggregate TCP flows," in *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, 2002, pp. 1–15.

[25] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," in *Proc. of the SIGCOMM '04*, 2004, pp. 281–292.

[26] R. J. Larsen and M. L. Marx, *Introduction to Mathematical Statistics and its Application; 4 edition.* Prentice Hall, 2005.

[27] K. Arvind, "Probabilistic clock synchronization in distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 5, pp. 474–487, 1994.

[28] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, "UMAC: Fast and secure message authentication," *Lec. Notes in CS*, vol. 1666, pp. 216–233, 1999.

[29] P. Rogaway, "UMAC Performance (more)." [Online]. Available: www.cs.ucdavis.edu/ rogaway/umac/2000/perf00bis.html

[30] N. Shah, "Understanding network processors," Master's thesis, University of California, Berkeley, September 2001.

[31] W. Feghali, B. Burres, G. Wolrich, and D. Carrigan, "Security: Adding protection to the network via the network processor," *Intel Technology Journal*, vol. 06, pp. 40–49, Aug. 2002.

[32] L. A. Sanchez, W. C. Milliken, A. C. Snoeren, F. Tchakountio, C. E. Jones, S. T. Kent, C. Partridge, and W. T. Strayer, "Hardware support for a hash-based ip traceback," in *2. DARPA Information Survivability Conference and Exposition (DISCEX II)*, 2001, pp. 146–152.

[33] D. L. Mills, "Network time protocol (version 3) specification, implementation," *RFC 1305, IETF*, Mar. 1992.

[34] H. F. Wedde, J. A. Lind, and G. Segbert, "Achieving Internal Synchronization Accuracy of 30 ms Under Message Delays Varying More Than 3 msec," in *WRTP99, 24th IFAC/IFIP Workshop on Real Time Programming*, 1999.

[35] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. of the OSDI*, Dec. 2002, pp. 255–270.

[36] Emulab - Network Emulation Testbed, "http://www.emulab.net," 2006.

[37] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking (TON)*, vol. 1, no. 4, pp. 397–413, 1993.

[38] C. V. Hollot, V. Misra, D. F. Towsley, and W. Gong, “On designing improved controllers for AQM routers supporting TCP flows,” in *Proc. of the INFOCOM ’01*, apr 2001, pp. 1726–1734.
[39] S. Athuraliya, S. Low, V. Li, and Q. Yin, “REM: Active queue management,” *IEEE Network*, vol. 15, no. 3, pp. 48–53, 2001.
[40] S. Floyd, “TCP and Explicit Congestion Notification,” *ACM Computer Communication Review*, vol. 24, no. 5, pp. 10–23, 1994.
[41] K. Ramakrishnan, S. Floyd, and D. Black, “The Addition of Explicit Congestion Notification (ECN) to IP,” *RFC 3168, IETF*, 2001.
[42] L. Le, J. Aikat, K. Jeffay, and F. D. Smith, “The effects of active queue management on web performance,” in *Proc. of the SIGCOMM ’03*, 2003, pp. 265–276.
[43] A. Kuzmanovic, “The power of explicit congestion notification,” in *Proc. of the SIGCOMM ’05*, 2005, pp. 61–72.
[44] K. Pentikousis and H. Badr, “Quantifying the deployment of TCP options - A comparative study,” *Communications Letters, IEEE*, vol. 8, no. 10, pp. 647–649, 2004.
[45] A. Medina, M. Allman, and S. Floyd, “Measuring the evolution of transport protocols in the internet,” *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, pp. 37–52, 2005.