情報処理学会論文誌 Vol. 49 No. 7 2634–2642 (July 2008)

推薦論文

Proactive DPD Trigger: IPsec SAの一貫性喪失状態からの効率的な回復方式の設計と実装

石山政浩[†1] 神田充[†1] 福本淳[†1]

本稿では，IPsecにおけるDead Peer Detection（DPD）を効率的に行う方法について提案する．IPsec Security Association（IPsec SA）の一貫性喪失からの回復にはDPDが有効である．しかし現在のDPDの方式として使用されているKeepAliveなどの方式では定期的な探査パケット交換による帯域の圧迫や，不必要な電力使用が発生し，センサネットワークなどでの使用には問題がある．本稿では，DPDを開始するタイミングを効率的に検知する方法について議論し，通信トラフィックを監視することによって能動的にDPDを行うProactive DPD Triggerの実装方式を提案する．また，Linux 2.6上で実装を行い，提案方式がIPsec SA一貫性喪失の検出に定期的な探査パケット交換を必要とせず，通信帯域を圧迫しないことを示す．さらに実装を評価することにより処理オーバヘッドが十分小さいことを示し，センサネットワークなどにおけるIPsecの使用にはProactive DPD Triggerが有効であることを示す．

Proactive DPD Trigger: Design and Implementation of an Effective Method for Recovering from IPsec SA Inconsistency

Masahiro Ishiyama,[†1] Mitsuru Kanda[†1] and Atsushi Fukumoto[†1]

We propose an effective method for Dead Peer Detection (DPD) activation in the IPsec protocol. Although DPD provides recovery from IPsec Security Association (IPsec SA) inconsistency, there are several issues to be addressed, especially for sensor-networks, such as exchanging unnecessary probe packets and superfluous power consumption. In this paper, we discuss methods that enable to determine a timing of performing DPD and propose an IPsec traffic-monitoring based DPD invocation mechanism, which is called "Proactive DPD Trigger." We implemented the mechanism on Linux 2.6, and we show that the mechanism does not need unnecessary probe packets. The performance evaluation of our implementation shows low processing overhead.

1. はじめに

近年，センサなど，建築物に付随するさまざまな機器（ファシリティ）をインターネットに接続し，活用しようという動きが高まっている．このような制御系ネットワークのIP化においては，情報セキュリティは重要な課題である．従来の制御系ネットワークは，仕様が公開されていないことをもって，一般のユーザが制御系ネットワークへ接続することを難しいものとして運用していた例が多い．一方で，制御系ネットワークがオープンプロトコルであるIPを利用する場合，一般ユーザが誤って接続し制御系ネットワークを混乱させたり，悪意を持つユーザによる攻撃が容易になったりするといった懸念がある．

制御系ネットワークのIP化におけるセキュリティ技術では，IPsec[6]が注目されている．制御系ネットワークのIP化においては既存の上位層プロトコルをIPへと移植する場合が多いため，上位層で独自にセキュリティメカニズムを規定し実装するにはアプリケーション開発コストや管理コストが高くなる可能性がある．よってIPsecには，パケットの認証と暗号化によるアプリケーション非依存な共通のセキュリティフレームワークとしての利用が期待されている．また8 bitや16 bitといった低コストな制御デバイスのIP化においても，IPsecを使用したセキュリティの確保に対する取り組みが行われている[8]．

しかし，現在のIPsecでは，IPsec Security Association（IPsec SA）の一貫性喪失からの回復に問題がある．たとえば，多数のセンサがデータを蓄積するサーバに対してデータを定期的に送る例を考える．ここでサーバが障害により再起動したとする．このときセンサ側はサーバに対するIPsec SAを維持しているが，サーバはすべてのIPsec SAを失っている．このため，センサがIPsecされたパケットをサーバに送り続けても，サーバ側では合致するSAがないため，このパケットは（多くの実装では）暗黙のうちに破棄される（Silently Discard）[⋆1]．このためサーバはセンサ情報を受け取ることができず，またセンサ側では，パ

†1 株式会社東芝研究開発センター通信プラットホームラボラトリー
Communication Platform Laboratory, R&D Center, Toshiba Corporation
本稿の内容は2007年7月のマルチメディア，分散，協調とモバイル（DICOMO2007）シンポジウムにて報告され，コンピュータセキュリティ研究会主査により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．
⋆1 cf. RFC4301 Section 5.2

ケットがすべて Silently Discard されるため，IPsec SA の共有状態が失われたこと（IPsec SA の一貫性喪失）を知ることができない．そしてセンサ側の IPsec SA の lifetime が切れるまでは，この状態を脱することができない．

この問題を解決する方法の 1 つに IKEv1 [1] や IKEv2 [5]，そして KINK [10] といった鍵交換プロトコルで規定されている Dead Peer Detection（DPD）がある．DPD による IPsec SA の一貫性喪失の検知そのものは非常に有効であるが，これにも課題は残されている．それは「いつ DPD を行うのか?」ということである．

IKEv1 [2] における DPD の方式としては，HeartBeat や KeepAlive によるものが RFC3706 [2] で提案されている．これらの方式では，通信相手の状態を知るまでの時間は探査パケット（probe packet）の間隔に依存する．よって，より早く IPsec SA の一貫性喪失の状態を検知したいのであれば，探査パケットの間隔を短くしなければならない．しかし，間隔を短くすることはトラフィックの増加につながり，これは以下の問題を引き起こす．

帯域の圧迫：特に多くのノードと通信するサーバのリンクや，センサなどが無線で接続されている場合，無線帯域を探査パケットで消費する．

電力の圧迫：バッテリなどで駆動しているノードでは，探査パケットの間隔が短くなると長い時間 dormant mode を維持できないためバッテリを多く消費する．

計算能力の圧迫：DPD の探査パケットは一般的に暗号化が必要なため，計算能力を余分に消費する．

以上の問題を回避するため，DPD を開始するタイミングを効率的に検知する方法（DPD Trigger Mechanism）が必要となる．本稿では，まず DPD Trigger Mechanism の方式として，通信相手からの通知をもとに DPD を行う Reactive DPD Trigger と，通信トラフィックを監視することによって能動的に DPD を行う Proactive DPD Trigger の 2 種類を提案する．そしてそれぞれの方式について議論し，Proactive DPD Trigger の利点について述べる．また我々が行った Proactive DPD Trigger の Proof of Concept 実装の実装概要および評価について述べる．

2. 提 案 方 式

本章では 2 つの DPD Trigger Mechanism の方式を提案し，それぞれについて議論する．

2.1 提案方式 1：Reactive DPD Trigger

RFC4301 では，受信したパケットに合致する SA を受信者が保持していなかった場合には対象のパケットは破棄することになっている．多くの実装ではこのときには暗黙のうちに破棄する（Silently Discard）が，一方で RFC2521 [4] では ICMP Security Failures Messages の中で BAD SPI が定義されている．これは受信者が受信した IPsec パケットに対して該当する SA を保持していない場合に送信者にエラーとして通知するメッセージである．よって以下の実装を行えば，IPsec SA の一貫性喪失の状態から迅速に回復できると考えられる．

(1) 受信したパケットに合致する SA を受信者が保持していなかった場合，送信元に対して ICMP BAD SPI Message（BAD SPI）を送信．
(2) BAD SPI を受信した場合には，使用した SA を設定した鍵管理アプリケーション（Key Management Application）に対して BAD SPI を受信したことを通知．
(3) 鍵管理アプリケーションには，BAD SPI を受信したという通知をもとに DPD を実行．

本稿ではこの方法を Reactive DPD Trigger と呼ぶ．

2.2 Reactive DPD Trigger の問題点

Reactive DPD Trigger は効率的ではあるが，多くの問題を内包している．以下問題点について議論する．

2.2.1 非公開性の問題

Firewall などでは，その存在自体をネットワークに対して非公開にしたい場合がある．このような場合では，BAD SPI を実装しているノードは攻撃者が IPsec されたパケットを使用してネットワークを探査する攻撃によって存在を暴露してしまうという問題がある．

2.2.2 DPD 探査パケットを送信させる DoS 攻撃の可能性

BAD SPI は受信者が暗号的に認証することができないため，攻撃者が BAD SPI を偽造することは容易である．一方，BAD SPI は DPD 探査パケットを送信させるトリガになる．よって攻撃者は，攻撃対象ノードに偽造した BAD SPI を送信することによって，攻撃対象ノードに多数の DPD 探査パケットを生成させることが可能となる．

2.2.3 DPD 探査パケットを受信させる DoS 攻撃の可能性

DPD を送信させる攻撃の応用として，DPD 探査パケットを受信させる攻撃が考えられる．たとえば VPN GW のように IPsec SA を多数に保持しているノードがあるとする．仮に攻撃者がこの VPN GW がどのノードと通信しているかを知ることができれば，攻撃者は VPN GW の発信元 IP アドレスを偽り，VPN GW の通信相手それぞれに BAD SPI を送信する．この結果 VPN GW は同時に多数の DPD 探査パケットを受信することになる．

2.2.4 BAD SPI を送信させる DoS 攻撃の可能性

BAD SPI の応答レートに制約がなければ，攻撃者は多数の虚偽の IPsec のパケットを攻

撃対象に送信することで，多数の BAD SPI を生成させネットワークを飽和させることができる．また，発信元 IP アドレスに対して応答レートに制約を課しても，攻撃者は発信元 IP アドレスを偽装することで多数の応答を引き出せる可能性がある．

2.2.5　Security Policy 処理の複雑化

IPsec の仕様では，あるノードに対して IPsec を行うか行わないかは Security Policy（SP）で定義される．仮にノード A とノード B がすべての通信を IPsec を使用して行うと定義した場合において，A と B が通信中にノード B が再起動したとする．ここでノード B がノード A からの IPsec されたパケットを受信した場合を考える．ノード B はこのパケットに対応する SA を保持していないので，ノード A に対して BAD SPI を応答しなければならないが，SP によればノード A に送信するためには IPsec を使用しなければならない．しかし IPsec を使用するためには鍵交換を行わなければならないことになる．ここで鍵交換を行うことを選べば，任意のノードからの虚偽の IPsec されたパケットによって鍵交換の処理を起動させる攻撃が成立する．一方で，BAD SPI を IPsec せずに送信する場合には，このパケットだけ例外処理を行わなければならない．この例外処理は BAD SPI の受信側であるノード A 側にも発生する．すなわちノード A はノード B からのパケットはすべて IPsec が適用されており，それ以外は受信しないという SP になっているが，BAD SPI だけはつねに例外として受信しなければならない．

2.3　提案方式 2：Proactive DPD Trigger

Reactive DPD Trigger では，通信相手からの反応（ICMP BAD SPI message）を待ってから DPD Trigger を起こしていたが，一方で自ノード内で完結して DPD Trigger を発行する方法も考えられる．これは一般的に IP での通信はトラフィックが双方向に流れる点を利用して，送信側の SA（outbound SA）を使用すれば短期的には受信側の SA（inbound SA）が使用される可能性が高いという点を利用する．

よって以下の実装を行えば，IPsec SA の一貫性喪失を検出できると考えられる．

(1)　鍵管理アプリケーションは，SA の交換が終了する際に inbound SA と outbound SA の組（SA Pair）を把握．
(2)　鍵管理アプリケーションは，inbound/outbound SA をカーネルに通知した後で，さらにカーネルに SA Pair の情報を通知．
(3)　カーネル側は，SA Pair の情報を記憶．
(4)　outbound SA が使われたときに，inbound のパケットが来ると期待される時間だけ待つタイマを設定．
(5)　inbound SA が使われたときにはこのタイマを解除．
(6)　このタイマが expire したときには，パケットを送信したにもかかわらず期待される応答が受信できていない状況であるので，カーネルは鍵管理アプリケーションに DPD を行うように通知．
(7)　鍵管理アプリケーションはこのカーネルからの通知を受け，DPD を実行．

本稿ではこの方法を Proactive DPD Trigger と呼ぶ．

2.4　Proactive DPD Trigger の問題点

Proactive DPD Trigger にも同様にいくつかの問題が内在する．以下この問題点について議論する．

2.4.1　IPsec 処理のパフォーマンスへの影響

Proactive DPD Trigger の問題は，入出力パケットの IPsec 処理の際に，DPD Trigger のための処理も行わなければならず，パフォーマンスの低下の懸念がある．

2.4.2　DPD Trigger の妥当性の問題

まず IP 通信自体は双方向とは限らないため，outbound SA を使用した後に必ずしも inbound SA が処理されるとは限らない．また，上位層が双方向通信であったとしても，IPsec の処理自体は上位層と独立であるため，ある SA Pair において，outbound SA が使用された後，必ず inbound SA が処理されると期待することはできない．たとえ上位層セッションが理想的な，「あるデータを送信し，その応答を返す」という形式のセッションであったとしても，「応答を返す」側にはその応答が送信されてはこない．よって，たとえ inbound SA が使われていない状況でも，DPD Trigger 発行する必要がない場合もあり，DPD Trigger を発行するタイミングの妥当性（以下これを「DPD Trigger の妥当性」と呼ぶ）の問題がある．

2.5　提案方式についての議論

Reactive DPD Trigger は通信相手からの通知によって DPD Trigger を発行できるため，パフォーマンスおよび Trigger の妥当性に優れているが，一方でさまざまなセキュリティ上の課題がある．これらの課題は注意深く実装すれば回避可能なものもあるが，非公開性の問題など，運用の観点から発生する問題点を含んでいる．また，Reactive DPD Trigger は通信相手が必ず BAD SPI を実装している必要があり，通信している双方のノードが実装していない限りそもそも両者とも DPD Trigger を使用することができない．

一方で Proactive DPD Trigger は，セキュリティ上の問題も少なく，通信している一方のノードだけが実装していても，実装しているノード側では DPD Trigger を使用できる利点

がある．このため，本稿では Proactive DPD Trigger を実装し，Proactive DPD Trigger のコンセプトが実際に動作するかどうかを確認する．

3. Proactive DPD Trigger の Proof of Concept（PoC）実装概要

今回の PoC 実装で使用した環境は以下のとおりである．

Operating System:

Linux 2.6.16

Distribution:

Debian GNU/Linux 3.1 (aka “sarge”)

Key Management Applicaiton:

“kinkd” of Racoon2 20060712 [12)]

これらに以下の拡張を行い，Proactive DPD Trigger を実装した．

3.1 kinkd からの SA Pair の登録

kinkd がカーネルに対して inbound/outbound SA の登録が完了した後に，SA Pair の情報を設定するための PF_KEY メッセージを拡張した．このメッセージは SADB_X_SET_DPDTRIGGER とした．さらに，現在の PF_KEY の（Linux 2.6.16 の）仕様では，1 つの PF_KEY メッセージに対して SA の情報は 1 種類しか通知できないため，SA Pair として Peer 側の SA 情報を通知する SADB_EXT_SA_PEER を新規に追加した．

3.2 IPsec トラフィックに対する状態遷移の定義

IPsec トラフィックに対する状態遷移に対して，以下の状態を定義する．

CONSISTENT　対向の SA が同期していると思われる状態

WATCHING　対向の SA と非同期の可能性がある状態

TRIGGERING　DPD Trigger を発行中の状態

TRIGGERED　DPD Trigger を発行後の状態

また，状態遷移には以下の値を使用する．

CUR　（CURrent time）：現在時刻

TWAT　（Trigger Watchdog Activated Time）：WATCHING に入る前に最後に outbound SA を使った時刻

ISALU　（Inbound SA LastUsed）：最後に Inbound SA を使った時刻

OSALU　（Outbound SA LastUsed）：最後に Outbound SA を使った時刻

AD　（Acceptable Delay）：outbound SA を使用してから，inbound SA を使用するまでの許容時間

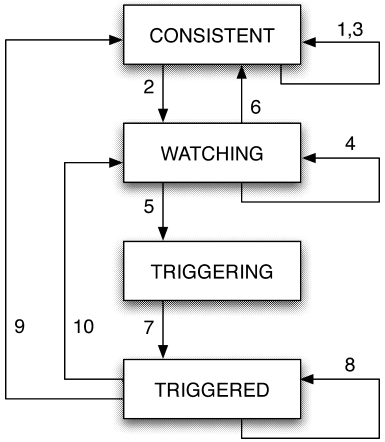

図 1　PoC 実装における状態遷移図：番号は状態遷移表と対応している
Fig. 1　State diagram in PoC implementation: the number in the diagram corresponds to the one in the state transition table.

GT　（Guard Time）：最後に inboud SA を使用した時間から，outbound SA を使用しても SA が同期していると見なせる許容時間

LTT　（Last Triggered Time）：DPD Trigger を発行した時刻

HDT　（Hold Down Time）：DPD Trigger を発行できる最小の時間間隔

状態遷移図を**図 1** に，状態遷移表を**表 1** に示す．

3.3 DPD Trigger の発行

カーネルが鍵管理アプリケーションに対して，PF_KEY を使用して DPD Trigger を通知するためのメッセージを追加した．このメッセージは SADB_X_DPDTRIGGER とした．メッセージの内容は SA expire と同様である．

鍵管理アプリケーションは，SADB_X_DPDTRIGGER をカーネルから受信した場合，SA expire と同様の方式で，その SA の交換を行った通信相手を特定する．その後，特定した通信相手に対して DPD を行う．

表 1　PoC 実装における状態遷移表：番号は状態遷移図と対応している
Table 1　State transition table in PoC implementation: the number in the leftmost column corresponds to the one in the state diagram.

#	State	Event	Action	New state
1	CONSISTENT	outbound SA を使用 ISALU + GT > OSALU		CONSISTENT
2	CONSISTENT	outbound SA を使用	TWAT=CUR	WATCHING
3	CONSISTENT	inbound SA を使用	—	CONSISTENT
4	WATCHING	outbound SA を使用 TWAT+AD < CUR	—	WATCHING
5	WATCHING	outbound SA を使用 TWAT+AD >= CUR	—	TRIGGERING
6	WATCHING	inbound SA を使用	—	CONSISTENT
7	TRIGGERING	—	DPD TRIGGER 発行 LTT=CUR	TRIGGERED
8	TRIGGERED	outbound/inbound SA を使用 LTT+HDT < CUR	—	TRIGGERED
9	TRIGGERED	inbound SA を使用	—	CONSISTENT
10	TRIGGERED	outbound SA を使用 LTT+HDT >= CUR	TWAT=CUR	WATCHING

4. Proof of Concept 実装の評価

4.1 Proactive DPD Trigger の挙動の確認と妥当性の評価

PoC 実装を用いて，既存のノードと IPsec を使用した通信を行い，パケットフローを観測することによって実装の挙動を確認した．パケットフローの確認には tcpdump を使用した．

なお今回の評価では，Acceptable Delay（AD）は 4 秒，Guard Time（GT）は 5 秒，Hold Down Time（HDT）は 60 秒とした．

4.1.1 DPD Trigger を使用した場合の TCP セッション確立のパケットフロー

DPD Trigger を使用した場合の IPsec 通信のパケットフローを**図 2** に示す．

I が Proactive DPD Trigger を実装した Initiator であり，R が通常のカーネルを使用した Responder である．1 度鍵交換を終了させてから，R 側で SA を消去し，R 上の鍵交換デーモンを再起動した．十分な時間が経過した後に，あらためて TCP のセッションを I から開始した．よってこの通信の開始状態においては I は R への送信側 SA を保持しているが，R はこの受信側 SA を保持していない．

```
01     0.000000    I > R: ESP(spi=0x09c2df8b,seq=0x2)     TCP SYN
02     3.171155    I > R: ESP(spi=0x09c2df8b,seq=0x3)     TCP SYN 再送 +3 秒
03     9.505683    I > R: ESP(spi=0x09c2df8b,seq=0x4)     TCP SYN 再送 +6 秒
04     9.506632    I.910 > R.910:  UDP, length:  652      DPD Trigger 発行：DPD 要求
05     9.508486    R.910 > I.910:  UDP, length:  132      DPD 応答
06    22.162371    I.910 > R.910:  UDP, length:  820      TCP SYN 再送 +12 秒を
                                                          trigger とした鍵交換要求
07    22.321980    R.910 > I.910:  UDP, length:  228      鍵交換応答
08    47.472291    I > R: ESP(spi=0x0ffc2349,seq=0x1)     TCP SYN 再送 +24 秒
09    47.473096    R > I: ESP(spi=0x09024c03,seq=0x1)     TCP ACK
```

図 2　DPD Trigger を使用した場合の TCP セッション確立のパケットフロー：03 のパケットによって DPD Trigger が発行され，鍵交換が行われる（06，07）．08，09 のパケットの SPI が異なっていることから，SA が更新されていることが確認できる
Fig. 2　Packet flow of TCP session initiation with the DPD Trigger mechanism: the packet 03 arouses DPD Trigger, and we can see the key exchange process (06, 07). We can also confirm that SAs are updated because SPIs in packets (08, 09) are different form the previous one.

01 は TCP SYN packet である．I は outbound SA を使用し，最後に inbound SA を使用した時刻（ISALU）よりも GT 秒以上経過していることから，CONSISTENT state から WATCHING state へと移行し，TWAT がこの時刻へと設定される．R 側では spi=0x09c2df8b に対応する SA が失われているため，このパケットは破棄される．02，03 も同様に破棄される．TCP exponential backoff による再送[9] のため，順次再送間隔が長くなっている．02 では，まだ TWAT から AD 秒経過していないため，WATCHING state を維持している．

03 のパケットで outbound SA を使用した時点ではすでに TWAT から AD 秒以上経過したため，TRIGGERING state に遷移する．この結果 DPD Trigger が発行され，鍵交換アプリケーションに DPD を促す．

04 が DPD 要求パケットである．DPD 応答が返され，I 側の SA が消去される（05）．次の TCP SYN の再送により，再度鍵交換処理が開始される（06，07）．現状の Linux 2.6.16（およびそれ以前）では，SA がない場合にはパケットは破棄されるため，この 06 を送信する原因となった TCP SYN パケットは送信されない．さらに次の TCP SYN が再送されると

(08)，新しい SA で I から送信されていることが SPI が異なっている（spi=0x09024c03）ことから確認できる．

以上より，Proactive DPD Trigger では，SA の一貫性喪失状態を迅速に検出し，能動的に鍵交換を開始できることが分かる．

4.1.2　定常状態における TCP セッションにおける Proactive DPD Trigger の振舞い

定常状態，すなわち SA の一貫性が維持できている場合の TCP セッションにおける Proactive DPD Trigger の振舞いについて考える．定常状態においては不要な DPD を行わないということが重要となり，これは Proactive DPD Trigger では CONSISTENT state を適切に維持できるかが鍵となる．

TCP においては，通信相手へデータを送信すれば，典型的にはそのデータに対する ACK が応答されるため，inbound SA が使用され，データ送信側は CONSISTENT state を維持できる．また，データ受信側は ACK を送信するため outbound SA を使用するが，これはデータ受信に対した時間（i.e. ISALU）から GT 秒以上後ということは考えにくいため，同様に CONSISTENT state を維持できる．

TCP セッションの終了時について考える．典型的には終了時にも 3-way handshake が行われるが，この場合ノードは FIN-ACK を出す側になるか，FIN-ACK に対する ACK を出す側のどちら側にもなりうる．提案方式では，FIN-ACK を出す側になった場合には FIN-ACK に対する ACK によって CONSISTENT state を維持できる．逆に FIN-ACK に対する ACK を出す側であっても，データ通信中と同様に，この ACK を出す時間が FIN-ACK を受信した時間（i.e. ISALU）から GT 秒以上後ということは考えにくいため，やはり CONSISTENT state を維持できる．よって，定常状態における TCP セッションでは不要な DPD を行わないということが分かる．

4.1.3　1 方向の通信における Proactive DPD Trigger のパケットフロー

Proactive DPD Trigger では，通信が双方向であることを仮定しているため，1 方向のパケット送信では正しく SA の一貫性喪失を検出することができない．このため，outbound SA を使用するごとに DPD を行ってしまう可能性もある．本実装では，TRIGGERED state を HDT だけ維持することにより，DPD Trigger を短い時間で発行することを避けている．

この動作を検証するために，10 秒に 1 回 UDP のデータを送信するプログラムを使用し，Proactive DPD Tigger によって DPD のパケットがどのように発行されるかを確認した．このときのパケットフローを**図 3** に示す．

```
01      0.000000   I.910 > R.910:  UDP, length:  652      DPD 要求
02      0.000973   I > R: ESP(spi=0x0725b646,seq=0x2)
03      0.001557   R.910 > I.910:  UDP, length:  132      DPD 応答
04     10.024912   I > R: ESP(spi=0x0725b646,seq=0x3)
05     20.039891   I > R: ESP(spi=0x0725b646,seq=0x4)
06     30.049878   I > R: ESP(spi=0x0725b646,seq=0x5)
07     40.070831   I > R: ESP(spi=0x0725b646,seq=0x6)
08     50.091112   I > R: ESP(spi=0x0725b646,seq=0x7)
09     60.108597   I > R: ESP(spi=0x0725b646,seq=0x8)
10     70.153662   I > R: ESP(spi=0x0725b646,seq=0x9)
11     70.154440   I.910 > R.910:  UDP, length:  652      DPD 要求
12     70.156287   R.910 > I.910:  UDP, length:  132      DPD 応答
13     80.146029   I > R: ESP(spi=0x0725b646,seq=0xa)
14     90.169481   I > R: ESP(spi=0x0725b646,seq=0xb)
15    100.211053   I > R: ESP(spi=0x0725b646,seq=0xc)
16    110.231456   I > R: ESP(spi=0x0725b646,seq=0xd)
17    120.250545   I > R: ESP(spi=0x0725b646,seq=0xe)
18    130.269537   I > R: ESP(spi=0x0725b646,seq=0xf)
19    140.304590   I > R: ESP(spi=0x0725b646,seq=0x10)
20    140.305376   I.910 > R.910:  UDP, length:  652      DPD 要求
21    140.306157   R.910 > I.910:  UDP, length:  132      DPD 応答
22    150.313039   I > R: ESP(spi=0x0725b646,seq=0x11)
```

図 3　1 方向の通信における Proactive DPD Trigger のパケットフロー：TRIGGERED state を HDT（=60）秒だけ維持するため，DPD 要求の間隔はそれ以上に保たれている

Fig. 3　Packet flow of unidirectional communication and DPD Trigger: the PoC implementation keeps TRIGGERED state for HDT (=60) seconds, thus the interval of DPD requests is kept in more than HDT seconds.

図 3 では，DPD Trigger を発行した状態から観測している．01 が DPD の要求であり，この DPD Trigger の発行を引きおこしたパケットは 02 である．DPD Trigger 発行後は HDT 秒間は TRIGGERED state を維持するため，DPD は以後しばらく発行されないことが分かる．今回 HDT は 60 秒に設定してあるため，09 のパケットによって WATCHING state へと戻る．その後，10 のパケットが送信されると，TRIGGERING state へと遷移し，DPD Trigger が発行され，DPD 要求が送信される（11）．受信側の SA は失われていないので，DPD が行われても SA が変化しないことは SPI が変化していないことから分かる（spi=0x0725b646）．

このように，短い間隔での 1 方向のみのパケット送信は，提案方式では DPD Trigger の発行頻度が最大となるが，Hold Down Time を導入したことによって，DPD 要求のパケットは抑制され，このように短い間隔でパケットを 1 方向に送出する場合においても定期的

表 2　性能計測に使用した機器仕様の概要
Table 2　Summary of the testbed of the performance evaluation.

	Device Under Test (DUT)	Tester
Processor	Pentium II (Klamath) 300 MHz	Dual-Core AMD Opteron 2214HE 2.2 GHz
Memory	64 MByte	8 GByte
Gigabit NIC	Intel 82545 (PCI)	Broadcom BCM5708 (PCI-X)

に KeepAlive を行う方式と変わらないことが分かる．

さらに，提案方式では outbound SA の使用，すなわちパケットの出力がなければ DPD は発行されない．よって，たとえアプリケーションが 1 方向性であっても，インターバルの長い間欠的な送信であれば，そのインターバルの間は DPD は発行されないため，定期的な KeepAlive よりも DPD のパケットは少なくなる．一方で，HDT 秒を超えていれば，パケットを送出した直後に DPD を起動できるため，より適切なタイミングで一貫性喪失の確認を行うことができる．よって，KeepAlive よりも効率的に一貫性喪失の検出が行えることが分かる．

4.2　IPsec 処理性能への影響についての評価

PoC 実装における Proactive DPD Trigger の処理が IPsec 処理性能に与える影響を評価した．

評価は Proactive DPD Trigger を実装した Device Under Test（DUT）と通常の Linux カーネルを使用した Tester を 1 Gbps Ethernet Switch を経由してそれぞれ 1000Base-T で接続して行った．DUT，Tester の機器仕様の概要を**表 2** に示す．

4.2.1　TCP バルク転送の性能

Proactive DPD Trigger の処理が，IPsec の処理性能に与える影響を測定するため，IPsec を使用した場合における TCP の最大バルク転送速度の比較を，DUT 上のカーネルを Proactive DPD Trigger を実装したもの（PoC 実装）と通常のカーネル（vanilla）との間で行った．TCP 性能の測定には iperf[11] を使用し，DUT がデータを送信する場合（generator）と受信する場合（sinker）に分けて計測を行った．

また，DPD Trigger を有効にした場合，IPsec の処理に対して必ず DPD Trigger の状態の操作が発生し，この操作は lock によって守られている．DUT は uniprocessor の構成のため，lock 処理の影響は軽微と予想されるが，一方で lock 処理の有無が Kernel Preemption[7] の効果に影響を与えることも考えられる．そこで本実装に由来する lock 処理の影響量を明らかにするため，Kernel Preemption を有効にしたもの（preemptive）と無効にしたもの（non-preemptive）それぞれについて測定した．また，netfilter は disable とした．結果を**表 3** に示す．計測結果は 20 秒間のデータ転送を 20 回行った平均（単位は Mbits/s）であり，括弧内はその標準偏差である．IPsec の mode は transport であり，ESP のみを使用した．AES-128 による暗号化（aes-128）と，暗号化を行わない場合（null）それぞれについて計測を行った．認証については双方とも HMAC-MD5 を使用した．また参考値として IPsec を使用しない場合の性能（IPsec なし）を計測した．loss の欄は vanilla に対する性能劣化率を%で示したものである．

表 3　TCP のバルク転送性能の評価：単位は Mbits/s. 括弧内は標準偏差．モードは ESP のみであり，aes-128, null はそれぞれ AES-128 と暗号化を行わなかった場合である．loss の欄は性能劣化率を%で示したもの
Table 3　TCP bulk transfer performance (in Mbits/s, numbers in parentheses are standard deviation): IPsec mode is ESP only, and aes-128 and null are using AES-128 encryption and no encryption, respectively. The loss column shows performance loss rate (in percent).

preemptive		aes-128	loss	null	loss	ipsec なし
vanilla						
	sinker	40.6 (0.092)		75.7 (0.89)		200.1 (4.7)
	generator	36.0 (0.046)		59.9 (0.050)		212.0 (0.43)
PoC 実装						
	sinker	40.5 (0.076)	0.25	75.4 (0.59)	0.40	
	generator	35.9 (0.051)	0.28	59.8 (0.060)	0.17	

non-preemptive		aes-128	loss	null	loss	ipsec なし
vanilla						
	sinker	38.0 (0.041)		75.3 (0.83)		207.3 (2.5)
	generator	36.2 (0.077)		60.2 (0.094)		218.8 (0.51)
PoC 実装						
	sinker	38.0 (0.054)	0.00	74.3 (0.86)	1.3	
	generator	36.1 (0.15)	0.28	59.9 (0.24)	0.50	

この結果より，TCP バルク転送の場合における性能劣化はほぼ測定誤差範囲内に収まっており，特に AES 暗号化のような重い処理を行っている場合には有意な差はない．AES の暗号化を行わない場合（null）にはわずかな処理劣化が観測できる．これはたとえば暗号化をハードウェアなどで処理した場合における Proactive DPD Trigger が IPsec 処理性能に与える影響がある可能性を示唆しているが，ほぼ無視できる範囲の性能劣化にとどまっているといえる．

よって Proactive DPD Trigger を実装した場合でも，IPsec 処理性能に大きな性能劣化をもたらすことはないことが分かった．また，Kernel Preemption を有効にした場合でも

表 4　OProfile を用いて測定した Proactive DPD Trigger の処理オーバヘッド：カーネルの総処理量に対する Proactive DPD Trigger の処理量の割合 (%)
Table 4　Measurement of kernel overhead of Proactive DPD Trigger process by OProfile: the percentage of Proactive DPD Trigger processing time in total kernel processing time.

	aes-128	null
sinker	0.43	0.72
generator	0.57	0.78

性能に変化がないことが確認された．

4.2.2　Kernel Profiling による処理オーバヘッドの計測

Kernel Profiler を用いて実際の Proactive DPD Trigger の処理のオーバヘッドを計測した．Profiler には OProfile[3] を用いた．計測方法は，iperf で 60 秒間 TCP のバルク転送を行い，Kernel Profiling を行った．得られたカーネル内全処理量から，Proactive DPD Trigger 処理のコールフローに含まれる処理量を抜き出し，その合計値を求めた．結果を**表 4** に示す．単位は%であり，これは，カーネルの総処理量に対する Proactive DPD Trigger の処理量の割合を示している．

この結果より実際の処理量は十分に小く，やはりオーバヘッドは十分に小さいことが確認できる．

4.2.3　複数の SA が設定されている場合における処理オーバヘッドの計測

これまでの計測は SA が Tester への 1 組しか設定されていない場合の評価であった．本項では，カーネル内に Tester 以外の SA を設定して，本実装のカーネル内の SA の設定量に対するスケーラビリティを調べる．

まず SA の組をそれぞれ 10 組，100 組，1,000 組と設定した場合の Kernel の処理オーバヘッドも計測した．Proactive DPD Trigger の処理オーバヘッドを変化を見るため，より相対処理量が多くなる暗号化を行わない場合（null）の場合について計測した．結果を**表 5** に示す．

また，SA の組を 1,000 組設定した状態で，TCP の最大バルク転送速度も行った．環境などは 4.2.1 項に準じる．Proactive DPD Trigger の処理オーバヘッドを変化を見るため，より相対処理量が多くなる暗号化を行わない場合（null）の場合について計測した．Kernel Preemption は無効とした．結果を**表 6** に示す．

これらの結果から，PoC 実装においては，Proactive DPD Trigger の処理オーバヘッドは，カーネルに設定されている SA の設定量の影響はほとんど受けないことが分かる．よって，SA の設定量に対するスケーラビリティがあることが分かった．

表 5　設定されている SA の量を変化させた場合における Proactive DPD Trigger の処理オーバヘッド：カーネルの総処理量に対する Proactive DPD Trigger の処理量の割合 (%)
Table 5　The percentage of processing overhead of Proactive DPD Trigger when the number of installed SAs is changed.

	1 組	10 組	100 組	1,000 組
sinker	0.72	0.68	0.72	0.74
generator	0.78	0.73	0.78	0.78

表 6　SA が 1,000 組設定されている状況下での TCP のバルク転送性能の評価：単位は Mbits/s．括弧内は標準偏差．モードは ESP のみであり，暗号化は行っていない．loss の欄は性能劣化率を%で示したもの
Table 6　TCP bulk transfer performance (in Mbits/s, numbers in parentheses are standard deviation) with preconfigured 1,000 SA pairs: IPsec mode is ESP only with no encryption. The loss column shows performance loss rate (in percent).

non-preemptive		null	loss
vanilla			
	sinker	75.2 (0.63)	
	generator	60.6 (0.20)	
PoC 実装			
	sinker	75.2 (0.60)	0.00
	generator	60.1 (0.12)	0.83

5. おわりに

本稿では，IPsec における DPD Trigger の必要性について議論し，現在我々が検討している Proactive DPD Trigger の Proof of Concept 実装について述べた．Proactive DPD Trigger は送信側 SA を使用した際に受信側 SA の利用を監視することにより，IPsec SA の一貫性喪失を検知可能であることを示した．また提案方式は IPsec SA の一貫性喪失の検出に探査パケットなどを使用しないため，通信帯域を圧迫せず，1 方向性の通信においても定期的な KeepAlive と変わらない結果となることを示した．また処理オーバヘッドは十分小さいことを示し，IPsec 通信における Proactive DPD Trigger が有効であることを示した．

今後は，マルチプロセッサ環境での性能に対する影響の確認や，多数の通信相手と実際に通信をした場合における性能評価などを行い，Proactive DPD Trigger が実環境での使用に耐えうることを検証していく．

参 考 文 献

1) Harkins, D. and Carrel, D.: The Internet Key Exchange (IKE), RFC 2409, IETF (1998).
2) Huang, G., Beaulieu, S. and Rochefort, D.: A Traffic-Based Method of Detecting Dead Internet Key Exchange (IKE) Peers, RFC 3706, IETF (2004).
3) John Levon: OProfile. http://oprofile.sourceforge.net/
4) Karn, P. and Simpson, W.: ICMP Security Failures Messages, RFC 2521, IETF (1999).
5) Kaufman, C. (Ed.): Internet Key Exchange (IKEv2) Protocol, RFC 4306, IETF (2005).
6) Kent, S. and Seo, K.: Security Architecture for the Internet Protocol, RFC 4301, IETF (2005).
7) Love, R.: Lowering latency in Linux: introducing a preemptible kernel, *Linux Journal*, Vol.2002, No.97, p.1 (2002).
8) Okabe, N., Sakane, S., Miyazawa, K., Kamada, K., Ishiyama, M., Inoue, A. and Esaki, H.: Implementing a Secure Autonomous Bootstrap Mechanism for Control Networks, *IEICE Trans. Communications*, Vol.E89-D, No.12 (2006).
9) Postel, J.: Transmission Control Protocol, RFC 0793, IETF (1981).
10) Sakane, S., Kamada, K., Thomas, M. and Vilhuber, J.: Kerberized Internet Negotiation of Keys (KINK), RFC 4430, IETF (2006).
11) The Board of Trustees of the University of Illinois: iperf. http://dast.nlanr.net/Projects/Iperf/
12) The Racoon2 Project: Racoon2 – a system to exchange and install security parameters for IPsec. http://www.racoon2.wide.ad.jp/

(平成 19 年 12 月 29 日受付)
(平成 20 年 4 月 8 日採録)

推 薦 文

センサネットワークにおいて，現在用いられている IPsec の Dead Peer Detection（DPD）における keep alive などの方式の問題点を改善し，効率化を図った Proactive DPD Trigger を提案し，実装評価している．提案方式は DPD の開始タイミングを検知し，通信トラフィックを監視することで，IPsec SA 一貫性喪失の検出に定期的な探査パケット交換を必要とせず，通信帯域を圧迫しない方式である．さらに，実装を用いた性能測定を通して提案方式の処理オーバヘッドが十分に小さいことを示している．課題設定ならびに方式提案/評価のいずれにおいても，参考となる論文であることから推薦する．

（コンピュータセキュリティ研究会主査　寺田真敏）

石山　政浩（正会員）

1994 年横浜国立大学大学院修士課程修了．現在，（株）東芝研究開発センター通信プラットホームラボラトリーに勤務．工学博士．Moblie IPv4 および IPsec の実装，IPv6 における移動透過保証方式の検討，IETF への標準化提案等を行う．主にネットワークレイヤを中心としたモーバイルコンピューティングおよびセキュリティ技術の研究開発に従事．

神田　　充

1999 年東北大学大学院情報科学研究科修了．現在，（株）東芝研究開発センター通信プラットホームラボラトリーに勤務．IPsec 等，コンピュータネットワークのセキュリティ技術に関する研究開発に従事．

福本　　淳

1966 年生．慶應義塾大学大学院理工学研究科計算機科学専攻修了．（株）東芝研究開発センター通信プラットホームラボラトリー勤務．主にコンピュータネットワークのセキュリティ技術に関する研究開発に従事．