# Pocket Bluff: A cooperation enforcing scheduled pocket switching protocol

Pars Mutaf

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *Pocket Bluff*

## *A cooperation enforcing scheduled pocket switching protocol*

Pars Mutaf

**N° 5664**

30 Août 2005

Thème COM


ISSN 0249-6399 ISRN INRIA/RR--5664--FR+ENG

# Pocket Bluff

## A cooperation enforcing scheduled pocket switching protocol

Pars Mutaf*

Thème COM — Systèmes communicants
Projet Planète

Rapport de recherche n° 5664 — 30 Août 2005 — 15 pages

**Abstract:** We propose POCKET BLUFF; a novel pocket switched message forwarding protocol that can foil selfish/malicious nodes that deliberately drop data. POCKET BLUFF hides the path of a message from intermediary nodes. It employs *hook* and *snake* shaped paths so that an intermediary node can 'also' be the final destination of a message. By deliberately dropping a message, a selfish/malicious intermediary node risks loosing incoming data.

We detail the POCKET BLUFF protocol, and its design rational step-by-step.

**Key-words:** Pocket switched networks, selfish/malicious behavior, cooperation enforcement, bluff, cryptography

* INRIA

# Un protocole de routage de poche en poche stimulant la coopération

**Résumé :** Ce document propose un nouveau protocole de "routage de poche en poche" qui fournit une défense contre les attaques de déni de service passif. Le protocole de routage que l'on propose cache le chemin de routage des utilisateurs intermédiaires. De plus, il utilise des chemins en forme de crochet et serpent. C'est-à-dire qu'un utilisateur intermédiaire peut également être la destination finale des données. Par conséquent, un utilisateur malicieux qui rejette des données (au lieu de les retransmettre) risque de perdre ses propres données.

**Mots-clés :** Réseaux "de poche en poche", déni de service passif, stimulation de la coopération, bluff, cryptographie

# 1 Introduction

Pocket switched networking (PSN)[3][4] is a recently brought up networking paradigm aimed at enabling network services for mobile users even when they are not in reach of Internet connectivity islands. In PSN, data is carried in one's trouser pocket, in a small wireless device e.g. a cellular phone, or PDA, from location to location, transfered to another user, which transports the data even farther, until it reaches the destination. Data transfer from pocket to pocket can generally be performed using wireless protocols e.g. 802.11 or Bluetooth that provide local connectivity. For example, Bob can transfer 1M data to Alice's PDA using 802.11 in ad hoc mode at disco, and Alice can transfer it to David (Alice's boy friend who stayed at home tonight), via Bluetooth. This protocol may be used if, for example, David has no Internet connection at home, because he never uses Internet at home and he prefers going to a cybercafé when absolutely necessary[1].

Two key words are generally associated with PSN: delay tolerance[2], and opportunism. PSN is only suitable for delay tolerant applications. Data is transported in user pockets in unpredictable ways and the data transport can have very *long latency*; i.e. until a mobile user meets the next mobile user. User mobility is a must for PSN, however its trajectory cannot be always controlled. At best, user mobility can be predicted to some extent, and this information can be used to select better ways to reach the destination. PSN is *opportunist* in that, it profits from *contact opportunities* that arise whenever mobile devices come into wireless range due to mobility of their users.

PSN is a new networking paradigm and opens a rich research area. In this paper, we explore a new garden of PSN research: denial-of-service and selfish behavior. We tend to believe that some pocket switching implementations will act selfishly, i.e. silently drop messages (e-mails, files) destined for others. They may do so for conserving energy, or memory[3] but also because they simply have no reason to help others. Most importantly, we expect malicious behavior due to, for example, jealousness. We imagine implementations that report:

```
143Kbytes of data is received. The data is
for someone else. Are you sure to keep it
in memory? [YES/NO].
```

By selecting NO, the user can deny service to the sender and receiver for a selfish reason. The user may also know the sender and/or receiver in advance and he may drop the file by jealousness or fear depending on the context.

The selfish routing behavior has received considerable attention in the context of mobile ad hoc networks ([6][7][2] to cite a few). In a mobile ad hoc network, all nodes behave as routers and take part in the discovery and maintenance of routes to other nodes in

[1] In support of this argument: the author is in full agreement with David's choice.

[2] Delay Tolerant Networking [5] is a more general networking paradigm that encloses PSN.

[3] Unlimited memory can generally be assumed. However, the question remains: *the fact that a resource is unlimited, implies that its owner is willing to share it?*

the network. This, however, assumes that nodes are willing to consume energy to forward packets destined for others. The common belief is that that some implementations will show selfish behavior. Selfish nodes will profit from the routing service offered by other nodes, but they will not provide the same service. I.e. they will silently drop packets destined for others. Different solutions have been proposed for detecting and penalizing selfish behavior. The detection part is, in general, observation-based and makes use of a tool called *watchdog*. Upon sending a packet, a watchdog-enabled node listens its neighbor's (i.e. the next node on the path to the destination) activity. If the neighbor has not retransmitted the same packet, then it is a selfish neighbor and it has probably dropped the packet for conserving energy. Based on the node behavior statistics, a reputation is assigned to each node. Nodes that have a bad reputation, i.e. selfish nodes, are penalized with similar behavior. The reputation information is also used for path rating and selecting better, i.e. more reliable, paths with more cooperative nodes.

In PSN, for example, Alice can forward a message to Bob in Paris, and Bob can forward it to Carol when they meet in a remote African village without Internet connection. Clearly, in such conditions, it is difficult to watch the next node's behavior in real-time. If Bob drops the data, Carol and Bob can detect it later. In this case, a bad reputation could be given to Bob (but, sometimes it may be too late since the data did not reach Carol in time). While we believe that observation and reputation-based cooperation enforcement techniques may be applicable to PSN, in this paper we do not discuss them. Their inventors are certainly better placed to make a decision.

Instead, in this paper we propose a radically different cooperation enforcement technique and we directly apply it to PSN. We do not deal with observations, nor reputations. We aim at preventing selfish/malicious behavior through bluff and cryptography.

## 2 Pocket Bluff

We have developed POCKET BLUFF; a cooperation enforcing unicast message forwarding protocol to be used on top of a pocket switched networking paradigm.

A *message* may be a text message, e-mail with or without attachments, file, etc. The size of a message is not specified, it depends on the user. It may be a simple message e.g. "Hi Bob try to call me...", or it can be a 1Mbyte audio file. The important point to note is that the "whole" message is forwarded at each hop. The source transfers the whole message to the next node (possibly over a reliable transport protocol), which transfers the message to a third node, etc.

POCKET BLUFF has three properties that make it a cooperation enforcing message forwarding protocol:

1. It hides the path of a forwarded message from intermediary nodes. Only the next node address is revealed to each node.

2. It employs hook and snake shaped paths so that an intermediary node may also be the final destination of a message. By deliberately dropping messages, a selfish/malicious intermediary node risks loosing an incoming message.

3. Only the final destination can read the message.

In the following sections, we describe the design and design rational of POCKET BLUFF, step-by-step. At time of writing, the details of PSN paradigm are yet to be explored and subject to future research. We should emphasize here that, POCKET BLUFF is applicable to what we call *scheduled pocket switching*; where the intermediary node identifiers are known by the source in advance. In other words, the user contacts needed for reaching the destination are planned or estimated before sending the message. With one or two intermediary nodes, this is a very reasonable assumption and can easily be achieved. However, contact scheduling may be difficult with an important number of contacts. In this paper we do not specify the contact scheduling issues (or, we assume a small number of hops i.e. contacts).

## 2.1 Hooks

The basic intuition behind POCKET BLUFF is illustrated in Figure 1. In the first example (Figure 1-a) the node $A$ is the source and the final destination of the message is $B$. A message can be sent by $A$ to $B$ via two different paths: $p1$ which is the direct path, or $p2$ via the next hop $C$ forming a hook-shaped path. If $p1$ is used, the final destination $B$ can immediately detect it and open the message (i.e. see and process its content). If, however, the node $B$ fails to open the message, then it faces the following possibilities:

- the message is for another node (and I am on its path and exploited for forwarding),
- the message is for me, but it has a hook-shaped path ($p2$), i.e. it will bounce me back from the next hop.

The goal of POCKET BLUFF protocol is to make it impossible for an intermediary node to answer the above question. In this case, the node $B$ will be obligated to forward $A$'s message because its path may be $p2$. The possibility that enforces each node to cooperate is marked above its name (Figure 1). The node $B$ cooperates because the message may follow the path $p2$ (the probability of $p2$ compared to $p1$ must be non-negligible). Similarly, the node $C$ must cooperate because the message may follow the path $p4$. The node $D$ must cooperate because a message may be received through the path $p6$.

## 2.2 Snakes

Hooks have a weakness. In a PSN environment, it may sometimes be easy to detect when data is being returned to the previous node; i.e. when the path is hook-shaped. For example, Alice sends data to Bob using a hook-shaped path via Carol's pocket. The data is first transfered to Bob. Then, Bob meets Carol, and transfers the data to her pocket. But the data was for Bob, and then Carol's maliciously programmed wireless device asks:

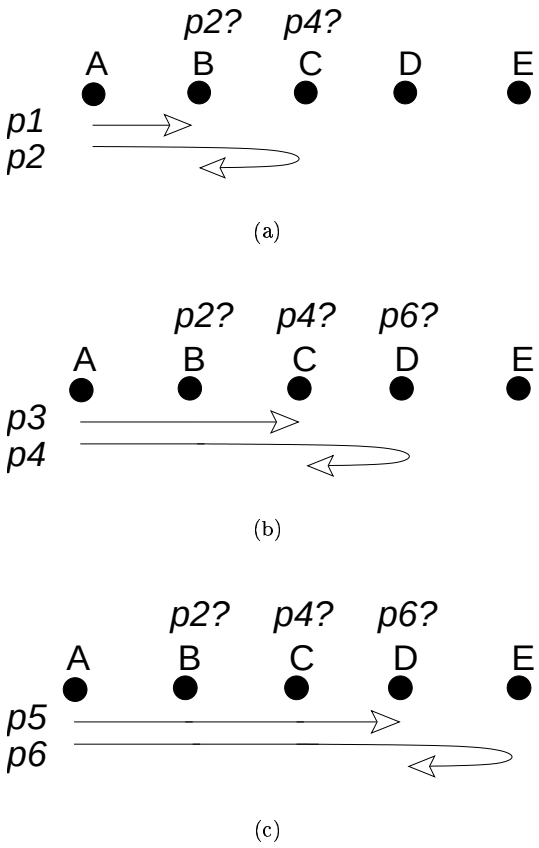


Figure 1: Hook-shaped paths.

```
You have received 56Kbytes of data. The
data is not for you. The presence of the
next node is detected. Are you sure to
forward it? [YES/NO].
```

In this case, Carol can understand that the data is for Bob, unless there are other possible candidates in range[4]. If Carol and Bob are alone, Bob is certainly the final destination of the data. In this case, Carol can select NO and destroy the data destined for Bob for any malicious reason.

[4]Carol's reasoning: The message is received when I have met Bob (i.e. the previous hop is probably Bob), and my PDA reports that the next hop is also Bob. This is a hook-shaped path. The data is not for me, it is for Bob.

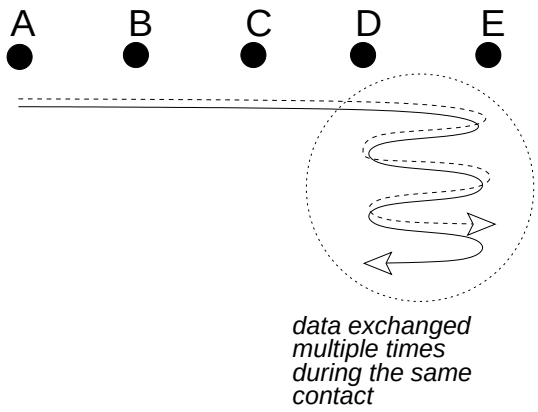


Figure 2: Snake-shaped paths. The data ping pongs a random number of times between the destination node (either $D$ or $E$) and the last node it has contacted. Note in this example that the nodes $B$ and $C$ cooperate because a ping pong might occur with the next node.

In Figure 1, it can be noted that this attack may be mounted by the node $C$ in Figure 1-a (if $p2$ is used), by the node $D$ in Figure 1-b (if $p4$ is used), and by the node $E$ in Figure 1-c (if $p6$ is used).

In order to address this problem, a snake-shaped path must be chosen by the source as illustrated in Figure 2. A snake-shaped path will ping pong a random number of times between the destination node and its neighbor[5], before arriving at the destination. The destination may be any of the two nodes i.e. snake-shaped paths should be used for both. Until the message reaches the destination, neither the destination nor its neighbor can determine the final destination with a probability greater than 0.5.

Let the node $A$ be the source and the node $E$ be the final destination of the message. The path of the message is chosen as usual, i.e. $B, C, D, E$. The intermediary nodes $B$ and $C$ are forced to forward the message as described above. Then, the source $A$ tosses a coin which we denote $r$. If the result is $r = 1$, then the path selection algorithm exits (with the output $B, C, D, E$). If, however, $r = 0$, then the path is appended two additional hops $D, E$. Until $r = 1$ is obtained, coins are tossed and the path is appended additional hops $D, E$.

It is important to note that the same algorithm will be employed for sending data to the node $D$ as well. The path will begin with $B, C, D$, coins will be tossed, and the path will be appended $E, D$ until $r = 1$.

Each time they receive the message, the nodes $D$ and $E$ face the following question:

-the message is "not" for me?

-or, coin toss gave $r = 0$ result?

[5]The term *neighbor* is used to mean the current user with whom there is PSN contact (e.g. a local 802.11 or Bluetooth connectivity).

which they cannot answer with a probability that is greater than 0.5. The expected cost introduced by a snake-shaped path (number of message ping pongs between the final two nodes) is denoted $s$ and equals:

$$s = \sum_{i=0}^{\infty} 2i \times (0.5)^i \simeq 4$$

I.e., snake-shaped paths will require on the average 4 times longer contact duration (between the last two nodes) than a simple message transfer without cooperation enforcement.

## 2.3 Protocol details

In this section we provide more detailed information on protocol operation, and tackle some performance issues.

One possible implementation of POCKET BLUFF can be built on a well-known construct called "onion" and public key cryptography[6]. Let $PK_i$ be the public key of node $i$, and $[data]_{PKi}$ denote that $data$ is encrypted with public key of node $i$.

A hook-shaped path such as $B, C, D, C$ can be constructed by the node $A$ as follows:

$$mid, DATA_K, [[[[K, eop]_{PK_C}, C]_{PK_{AD}}, D]_{PK_C}, C]_{PK_B}, B$$

where
- $DATA_K$ is the data encrypted with the secret key $K$,
- $mid$ stands for message identifier,
- $eop$ stands for "end-of-path".
- The rightmost protocol field (initially $B$) is the next node address.

Upon contact, this message is sent by $A$ to $B$. The node $B$ decrypts (or, peels) the outer onion layer with its private key and obtains

$$mid, DATA_K, [[[K, eop]_{PK_C}, C]_{PK_D}, D]_{PK_C}, C$$

The node $B$ obtains $C$ as next node (instead of $eop$). Then, upon contact with $C$, the message is sent to $C$ which decrypts the outer onion layer and obtains

$$mid, DATA_K, [[K, eop]_{PK_C}, C]_{PK_D}, D$$

Similarly, the node $C$ is not the final destination and sends the message to the indicated address $D$ upon contact. The node $D$ decrypts the outer onion and obtains

$$mid, DATA_K, [K, eop]_{PK_C}, C$$

[6]To our knowledge the idea is first used in Onion Routing[1]. The goal of onion routing is to protect the privacy of the sender and recipient of a message, while also providing protection for message content as it traverses a network (generally Internet). In POCKET BLUFF, we fully profit from these features of the onion structure. However, by using hook and snake-shaped paths, the novel feature of POCKET BLUFF, we also enforce cooperation to forward data.

This message is sent back to $C$ during the same contact. The node $C$ obtains

$$mid, DATA_K, K, eop$$

and decrypts $DATA_K$ with the secret key $K$.

A snake-shaped path can be constructed in a similar way.

It can be noted that, the above protocol avoids the processing cost of public key encryption/decryption. The data, which can be very large, is encrypted with a symmetric key $K$ enclosed in the onion. Only the final destination can find the symmetric key, hence decrypt the data. Public encryption/decryption is only performed for routing related information e.g. next hop addresses, and for the secret key $K$.

## 2.4 Optimization for fat snakes

We return our attention to snake-shaped paths. One problem is that the same data must be exchanged on the average four times between the final two nodes. This may be considered harmful if the is data very long. Exchanging the same data several times is energy consuming, and it requires on the average four times longer contact durations (for the last two nodes). The cost of the message ping ping can be very much reduced by separating the onion from the data. This is possible due to the indirection described above: data is encrypted with a symmetric key that is enclosed in the asymmetrically encrypted onion.

In the proposed optimization, the onion is detached from the data when all nodes receive a copy of the message. Then, the last two nodes will start to exchange and decrypt the onion, until one of them obtains the symmetric key used to encrypt the data. The other node will time out and delete the data since it is unreadable.

This procedure is illustrated in Figure 3. First, note that a reliable transport protocol is used to transfer large data. A different and independent connection is opened at each hop to forward the large data. Each intermediary node stores a copy of the forwarded data, hoping that the onion will be returned from the next hop. When the two final nodes receive a copy of the data, the onion is detached from data. At this point, only the onion is forwarded on the snake-shape path. Each time the onion is received, the outer onion layer is decrypted and sent to the other node, until one of the nodes obtains the symmetric key.

# 3 Security discussions

### 3.0.1 Hooks versus snakes

As mentioned above, hook-shaped paths may suffer from a weakness and reveal the final destination of a message. Consequently, we conclude that snake-shaped paths must always be used, unless a special case is found to be secure with hook-shaped paths.

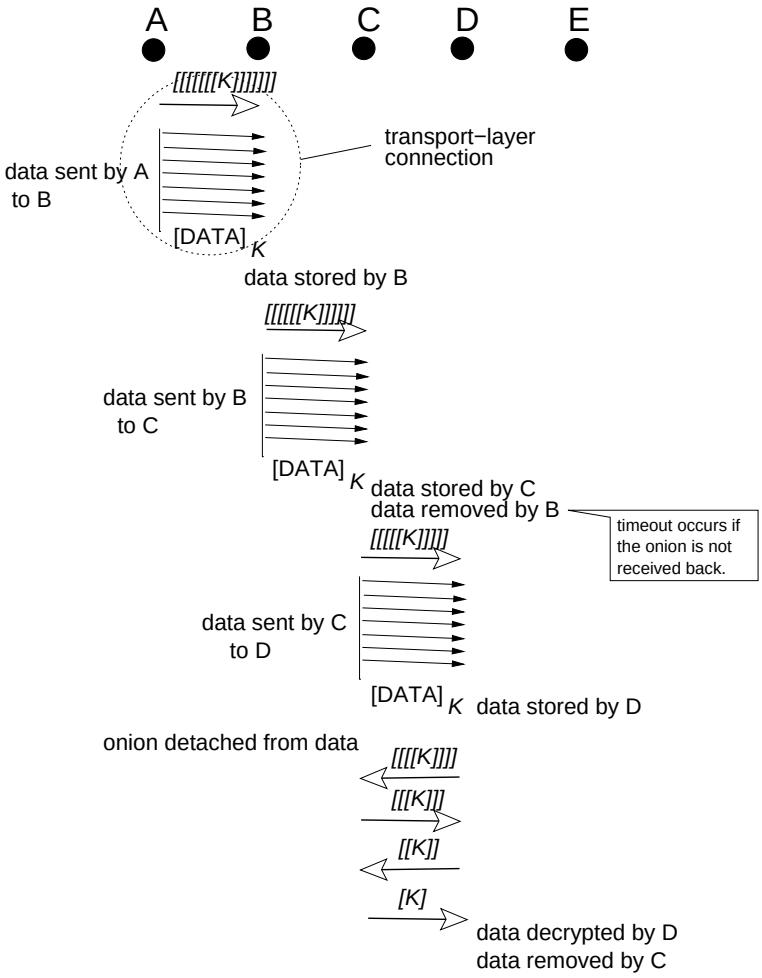


Figure 3: Optimized snake path for long messages.

### 3.0.2 Public keys

Each onion layer is encrypted with the corresponding intermediary node's public key. In this paper, we do not address the public key distribution issues. We assume that the public keys are authentic, and certified by a trusted authority.

Public keys are preferred because in PSN context the nodes are disconnected and hence secret key agreement may be difficult and take very long time. Another possible advantage is that public keys are not dependent on the source, and hence the source is not obligated to

reveal its identifier. User identifiers (sender and recipient) will be found in the data, possibly in standard e-mail format, and only accessible to the final destination.

One drawback of public key cryptography is its high processing cost. However, the onions are relatively small. Assuming and 32-bit IPv4 node identifier, and 128-bit hash of the data (see Section 3.0.5), we expect 160 bits per onion layer, i.e. per intermediary node (or, 128+128=256 bits with IPv6). Some padding will also be needed to hide the number of hops to the destination (See Section 3.0.3). The data itself can be very large, in the order of several Mbytes. The related processing cost is reasonable since data is encrypted with a symmetric key found in the onion.

### 3.0.3 Onion size padding

It can be noted that the onion size is reduced each time an intermediary node decrypts the corresponding onion layer and removes its address. The onion size must not reveal the number of hops to the final destination. For this purpose, additional onion layers can be added into the onion's center. Or, a random sized padding can be enclosed along with the symmetric key $K$ found at the onion's center.

We assume that the onion size (bits) divided by the onion layer size (bits) gives the number of hops to the destination. Using this information a selfish/malicious node can find out when it is not the final destination of a message. For example, we imagine that a malicious node receives a message, decrypts the outer onion layer, the data is still unreadable, but the destination is one hop away. In this case, the message can be maliciously dropped since it is either for the next node or the previous node. This problem can be addressed by adding additional but useless onion layers, or padding, to the onion's center.

Another problem occurs when the number of hops to the destination is very large. In this case, the first nodes on the path can detect that the destination is far away, i.e. the message will never come back. In this case the message can be maliciously dropped. This problem can also be solved with padding. Sometimes, unnecessarily long padding can be enclosed into the onion, creating doubt about the real distance to destination.

### 3.0.4 MAC/IP addresses

While we do not recommend hook-shaped paths, the following discussion is noteworthy:

The MAC and/or IP address may reveal it when the path is hook-shaped, i.e. *when the next hop address is equal to the previous node address.* In this case, the data can be selfishly dropped since it is for the previous hop. If needed, this problem can be addressed by configuring a random IP/MAC address when forwarding the message to the next node.

Note however that snake-shaped paths do not suffer from such problems. This is a second justification for preferring snakes to hooks.

### 3.0.5 Onion/data binding

A denial-of-service attack can be mounted against our optimization for fat snakes (Section 2.4). An intermediary node can store the original file, but forward bogus data. As a consequence, the malicious intermediary node will be able to read the data if it is the final destination (after exchanging/decrypting the onion several times with the next node, as described above). However, if the attacker is not the final destination, the real destination will be denied service; i.e. it will have the original secret key to decrypt the data, but the data itself is useless.

This problem can easily be addressed by inserting a cryptographic hash, e.g. SHA, of the data at each onion layer. In this case, upon decrypting its onion layer, an intermediary node can compare the hash value found in its onion layer against the hash of the received data. If they do not match, then the protocol can be stopped, preventing the attacker from receiving the symmetric key if he is the final destination.

## 4 Conclusion, future work, and challenges

We have proposed POCKET BLUFF; a cooperation enforcing pocket switching protocol. The POCKET BLUFF protocol, sometimes, in order to bluff the intermediary nodes, sends the data one hop farther than a message's real destination. The data is returned back to the real destination (via a hook or snake shaped path). Therefore, with POCKET BLUFF, sometimes, time will pass until an additional and useless contact occurs before data arrives at the destination.

Hoping that a message will be returned back from the next hop, intermediary nodes forward messages the indicated next node, i.e. cooperate to the pocket switching process.

The quality, or level, of cooperation enforcement depends on user. I.e. there is a non-negligible human factor in POCKET BLUFF, unless future work brings intelligent algorithms. For example, Alice, Bob and Carol form a pocket switched network, but none of them is willing to carry data for the two others. POCKET BLUFF can be used to enable pocket switched networking in this situation. However, care should be taken. For example, if Alice and Bob always communicate with each other via Carol's pocket, but they never send messages to Carol, then Carol will stop cooperating (unless Carol is waiting for a very important but lowly probable message from Alice or Bob). In POCKET BLUFF, each user must be convinced that cooperation will bring some benefit, i.e. a message will sometimes be received from the next hop. It is the users' role to convince other users that cooperation is in their interest, by creating different and interesting associations. These issues, however, need more rigorous analysis and left to future work.

In this paper, we have addressed the cooperation enforcement problem in *scheduled* pocket switching. The source node knows in advance the exact succession of the user contacts that will lead the data to the destination. Simple forms of scheduled pocket switching i.e. with one or two intermediary nodes are realistic and feasible. Contact scheduling may become more difficult with important number of intermediary nodes, however.

Our *scheduled* pocket switching assumption is also very in-line with the cooperation enforcement quality of our approach. Let $n_0$ be the source node, $n_i$ the current node (the node that carries the message), and $n_{i+1}$ the next node. I.e., $n_i$ will forward the message to $n_{i+1}$ upon contact. In our approach the user contacts are planned/estimated in advance. This necessarily implies that the source $n_0$ knows both $n_i$ and $n_{i+1}$ (since the source has estimated/planned the contact). The message may be for $n_i$ or $n_{i+1}$, since the source knows both of them. Consequently, a malicious/selfish reasoning e.g. "*I don't know the next node, this message is certainly not for me, I can drop it*" is naive.

Inspired from [5], in future work we will tackle the e-mail connectivity problem in a remote village without Internet connection. We imagine a cooperative approach. Alice (a villager), when it is her turn, goes by car to the city Internet connection point and checks e-mail for herself and other villagers [7], brings back the e-mails in her PDA or laptop and distributes them to their respective destinations in the village via, for example, a local wireless network. Thanks to this cooperation, the villagers can save gasoline.

Nevertheless, malicious/selfish behavior can also be expected in this scenario. The e-mail carrier, Alice or someone else, can selectively delete e-mails destined for other villagers for any malicious reason. POCKET BLUFF can defeat such denial-of-service. With POCKET BLUFF, an e-mail received from the Internet may (or may not) be returned back to Alice from another village user, through a hook or snake shaped path. This abstract representation is illustrated in Figure 4.

We give here a simple example with two villagers, Alice and Bob, to be developed in future work. For editorial simplicity, we let the whole data be encrypted with public keys. First, Alice ($A$) and Bob ($B$) send each other's identifier (and public key) to their e-mail correspondents. David is Alice's boy friend (and Bob does not like him). When David wishes to send an e-mail to Alice, he encrypts his message with Alice's public key $PK_A$, then Bob's public key $PK_B$, again with $PK_A$ etc. The result is an onion structure forming a snake-shaped path:

$$[[[[message]_{PK_A}, A]_{PK_B}, B]_{PK_A}, A]_{PK_B}, B$$

The resulting message is sent by David to the villager's common e-mail address via *anonymous* e-mail. Anonymous e-mail must be used to thwart an attack e.g. "*I don't know/like the sender of this e-mail, it is certainly for someone else, I can/should delete it.*". Then, Alice takes the village car, goes to the city and checks e-mail. The above message is unreadable for Alice and she brings the message to the village. The message is sent to the next node i.e. Bob, ping pongs a random number of times between Alice and Bob, and finally Alice obtains the message in clear-text. The message was for Alice (but it could be for Bob).

Alternative story:

Bob takes the village car, goes to the city and checks e-mail. The next hop is Bob and he can decrypt the outer onion layer. At this point however, Bob is in the same situation

[7] The village people possibly share the same Internet account and e-mail address, so that any user when it is his/her turn can check e-mail for every other user. E-mail confidentiality, in this case can be achieved by encrypting the messages.

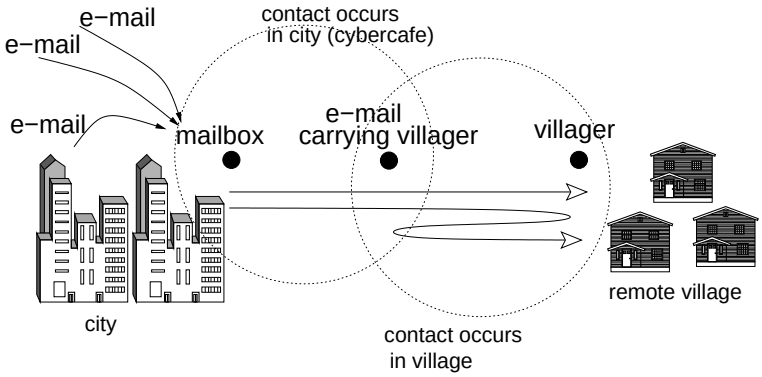


Figure 4: Cooperation enforcement in the remote village scenario.

as Alice in the above version of the story. He has an unreadable message, and brings it to the village. The message is sent to the next node i.e. Alice, ping pongs a random number of times between Alice and Bob, and finally Alice obtains the message in clear-text. The message was for Alice (but it could be for Bob).

Finally, we believe that the same technique (i.e. Bluff) can be applicable to other wireless networking cases such as multi-hop wireless networks and mobile ad hoc networks. In future work we will investigate these possibilities. In particular, we are planning to adapt the Bluff protocol to a wireless ad hoc short messaging system using secret (pairwise) or public keys.

**Acknowledgements**

The author would like to thank Refik Molva and Claude Castelluccia for discussions on this paper.

# References

[1] Onion routing (Wikipedia). URL: http://en.wikipedia.org/wiki/Onion_Routing.

[2] S. Buchegger and J. Le Boudec. Performance Analysis of the CONFIDANT Protocol: Cooperation of Nodes Fairness in Dynamic Ad HocNeTworks. In *Proceedings of IEEE/ACM Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc'02)*, Lausanne, Switzerland, June 2002.

[3] A. Chaintreau, P. Hui, J. Crowroft, C. Diot, R. Gass, and J. Scott. Pocket Switched Networks: Real-world mobility and its consequences for opportunistic forwarding. University

of Cambridge, Computer Laboratory Technical Report UCAM-CL-TR-617, February 2005.

[4] P. Hui, A. Chaintreau, J. Scott, J. Crowroft, R. Gass, and C. Diot. Pocket Switched Networks and the Consequences of Human Mobility in Conference Environments. In *Workshop on Delay Tolerant Networking*, Philedelphia, PA, August 22-26.

[5] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *SIGCOMM'04*, Portland, Oregon, August 30-September 3.

[6] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating Routing Misbehavior in Mobile Ad Hoc Networks. In *Proceedings of MOBICOM 2000,*, pages 255–265, Boston, Massatchusetts, August 2000.

[7] P. Michiardi and R. Molva. CORE: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In *Communication and Multimedia Security (CMS'02)*, Portoroz, Slovenia, September 26-27 2002.

# Contents

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399