1

On a QoS Intrusion Tolerant Routing protocol in Ad-hoc Networks

Neïla Krichene, Noureddine Boudriga

CNAS, University of Carthage, Tunisia, k.neila@voila.fr, nab@supcom.rnu.tn

Summary. We propose in this paper a protocol, called *QITAR,* which will secure the ad-hoc routing process while guaranteeing end-to-end QoS requirements in terms of delay or bandwidth constraints. Our approach provides an intrusion tolerant environment and uses and enhances the concept of *Trusted Timely Computing Base* in order to deduce consistent delay information and verify the bandwidth value pretended by untrusted nodes. *QITAR* also proposes a rescue procedure that saves the resources and accelerates the route maintenance in case of node mobility.

Intrusion tolerance, Ad-hoc Routing, QoS.

1.1 Introduction

A Mobile Ad-hoc NETwork is made up by several wireless mobile nodes which are temporary connected by multi-hop links. The nodes can freely move and form a changing topology. Consequently, the nodes should function as hosts and routers in order to communicate. An efficient routing protocol is vital to the deployment of performing MANETs, especially when it is required to provide intrusion tolerance and guarantee a specific level of quality of service (QoS). Most existing routing protocols for MANETs such as *AODV* [1], *DSR* [2] and *DSDV* [3] tried to cope with the variable nature of the network topology by providing a best effort service without considering relevant security issues. However, the dynamic topology makes it hard to keep consistent routing information and detect behavior anomalies. Furthermore, the lack of fixed infrastructure prevents first-line defenses. Meanwhile, the variable capacity, the energy constraints and the open environment increase the vulnerability of nodes and wireless channels; thus resulting in severe security problems.

On the other hand, the emergence of multimedia applications has resulted in a growing need of providing QoS in the MANETs context. The routing protocol should not only find a route but also satisfy the end-to-end QoS requirements, which are often given in terms of delay and bandwidth. As

a QoS route is expected to be robust and efficient, the intrusion tolerant property may be viewed as an additional QoS parameter that should be taken into account while searching for a suitable path. Unfortunately, the research activities have been conducted so far, guarantee either intrusion tolerance or QoS since the two properties are considered as conflicting. In fact, intrusion tolerant routing protocols try to secure the route establishment and data transfer at the extent of QoS guarantee while QoS routing protocols ignore security issues since they are resource and time consuming.

To have a complete vision of the routing problem, security requirements and QoS provision should be supported by the same scheme. Our main objective is to design an intrusion tolerant routing protocol for ad-hoc networks while guaranteeing QoS requirements. Our approach is to secure the routing protocol phases while guaranteeing the delay and bandwidth required by the requesting node. The intermediate mobile nodes will cooperate to the routing process; but they will be permanently supervised in order to detect the misbehaving. We will also use the concept of Trusted Timely Computing Base ($TTCB$) defined in [4] and enhance it for need of QoS provision and security. Our contribution in this paper is four-fold. First, we have proposed a new routing protocol called ***QITAR*** that provides accurate QoS while resisting to malicious attacks. Second, we have enhanced the $TTCB$ security services so that wormhole attacks and denial of service intrusions are tolerated. Third, we have taken advantage of the $TTCB$ reliable time services in order to secure the estimation of the transmission delays. Besides, we have used the estimated delays to verify the accuracy of the bandwidth value pretended by potentially malicious mobile nodes. Finally, we have proposed a rescue procedure that saves the resources and accelerates the route maintenance in case of nodes mobility.

The remainder of this paper is organized as follows. Section 1.2 summarizes the related work in the secure routing and the QoS routing for MANETs. In Section 1.3, we describe our proposed protocol, and in Section 1.4 we detail its design. Section 1.5 develops the proposed secure estimation of the delay and bandwidth. Finally, we discuss in Section 1.6 the security features of our protocol and we show how it is designed to tolerate serious attacks.

1.2 Related Work

Most of the researches in the ad-hoc communication field have addressed separately the routing security and the end-to-end QoS provision. In this section, we consider some existing protocols and discuss some of their shortcomings in order to palliate them while designing our scheme.

1.2.1 Secure routing

Although routing in ad-hoc networks is more vulnerable due to the wireless environment characteristics, many of the encountered security threats are sim-

ilar to those faced by wired networks. A malicious mobile node may cause a denial of service or attemp a wormhole attack [15]. It also can impersonate other nodes, spread false routing information, or drop all the packets passing through it. The existing secure protocols are either an extension of existing protocols such as *SAODV* [6], a set of mechanisms that can be defined upon any routing protocol such as *SRP* [7], or an independent new routing protocol that addresses some security issues like *ODSBR* [8].

The secure version of *AODV* (*SAODV*) computes digital signatures to authenticate the non-mutable fields of the signaling messages. Furthermore, it computes hash chains to secure the varying values. The resulting information is transmitted with an *AODV* message as an extension that is referred to as *Signature Extension*. *SAODV* is still a work in progress, its authors are currently trying to reduce the processing power requirements due to the use of asymmetric cryptography [6].

A different approach has been proposed by the *SRP* protocol and consists in mitigating the attacks of malicious nodes and guaranteeing the acquisition of correct topological information [7] while integrating mechanisms that protect the network functions against attacks exploiting *SRP* itself. The authors assumed the existence of a *Security Association* between the source and destination. Moreover, malicious nodes are assumed to exhibit arbitrary Byzantine behavior. *SRP* is able to operate without the existence of an on-line certification authority [7]. However, *SRP* does not guarantee the authenticity and integrity of the vital route error messages; thus enabling an attacker to harm the route it belongs to.

To resist to byzantine failures caused by individual or colluding nodes [8], *ODSBR* trusts only the source and the destination nodes and authenticates any intermediate mobile node (*MN*), while it expects the *MNs* to exhibit a byzantine behavior alone or in collusion with other nodes. Protocol *ODSBR* proposes an interesting procedure of detecting faulty links which are avoided in the process of route discovery. It is made up of three successive phases: 1) the *Route Discovery*; 2) the *Byzantine Fault Detection,* and 3) the *Link Weight Management* and employs on-demand shared keys to secure communications.

In this paper we propose an intrusion tolerant routing protocol that resists to common and serious attacks while securely estimating and providing the required QoS.

1.2.2 QoS Routing

Wireless ad-hoc networks have a fast-changing topology which influences the load conditions and the connectivity of the nodes. This dynamic nature makes it difficult to adapt nodes to changing conditions and monitor connection states and reservations. For all these reasons, supporting real-time and advanced applications with constraining QoS requirements in MANETs is considered as a challenging issue. Moreover, most of the existing protocols such as QoS for *AODV* [9] and *SWAN* [10] face important security threats.

The authors of the protocol *AODV* have specified extensions which can be used to guarantee a maximum delay and a minimum bandwidth along a route [9] but they did not protect the route against attacks. In fact, a mobile node independently computes its own delay and available bandwidth; hence it may make believe erroneous values of the information it sends in order to cause a denial of service or exhibit a selfish behavior. Moreover, any intermediate node can violate the integrity of the control packets. Finally, each node knows only its neighbors so it can not verify the behavior of the other members of the route it belongs to.

To address this issues, a maximum delay and a minimum bandwidth thresholds can be stated; thus obliging attackers to provide a minimum level of QoS. Furthermore, a cryptographic scheme must be utilized to provide integrity. Finally, a source node which does not receive the data packets acknowledgments should be able to ask its immediate neighbor for its signed delay value, the neighbor should ask its successor for the same data. This procedure will allow the source comparing the pretended QoS values to the QoS requirements it needs and detecting malicious nodes.

SWAN is a stateless network model which uses distributed rate control and feed-back control mechanisms in order to provide soft real-time services and service differentiation by regulating the admitted traffic in case of topology and QoS changes [10]. *SWAN* uses "probing" to obtain the minimal bandwidth available on a path. The admission control decision is only taken at source nodes. To regulate real-time sessions, each node continuously (and independently) measures the utilization of its traffic and starts marking the *Explicit Congestion Notification (ECN)* bits in the *IP* header of the real time packets whenever it detects serious violations. The destination monitors the *ECN* bits and notifies the source which may re-establish a new real-time session or terminate the pending one, if the QoS requirements can no longer be met. As other QoS routing protocols, *SWAN* did not address security issues. In fact, when a node wants to determine the delay on the link between it and its neighbor, it may accept false information and back off its rate causing best effort traffic starvation. Moreover, some attacking nodes will not correctly probe the network before admitting new real-time sessions; thus inducing a denial of service. Finally, attackers may corrupt the exchanged control packets or impersonate nodes.

As some malicious nodes may detect long delays but refuse to regulate their best effort traffic, a minimum threshold value $r_{\mathrm{BE}_{min}}$ should be specified so that the well-behaving neighbors can back off their best effort traffic rate until they reach the specified threshold. Moreover, a black-list containing the source of congestioned traffic should be defined at every intermediate node so that attackers can be isolated for awhile. In addition, a cryptographic scheme should be adopted in order to guarantee the authenticity and integrity of the exchanged control and data. Finally, a maximum threshold of session-re-establishments should be set at each source so that malicious nodes, which arbitrarily mark *Congestion Experienced* packets, cannot cause a denial of

service. Our work in the following is the design of a routing protocol that provides the aforementioned requirements, while considering the security as an additional QoS parameter that should be guaranteed.

1.3 The QITAR protocol: a QoS intrusion tolerant routing protocol

Our proposal will combine intrusion tolerance and QoS provision in one protocol called ***QITAR*** (*Qos and Intrusion Tolerant Ad-hoc Routing*). As mentioned earlier, ***QITAR*** uses a *Trusted Timely Computing Base* [4] modified version in order to detect the malicious behaviors and accurately meet the QoS requirements.

1.3.1 TTCB concept

The *TTCB* can be defined as a secure real-time distributed component which provides a set of trusted services related to time and security such as the trusted block agreement, trusted duration measurement, and trusted absolute timestamping [4]. The architecture of a system with a *TTCB* is suggested in Figure 1.1.

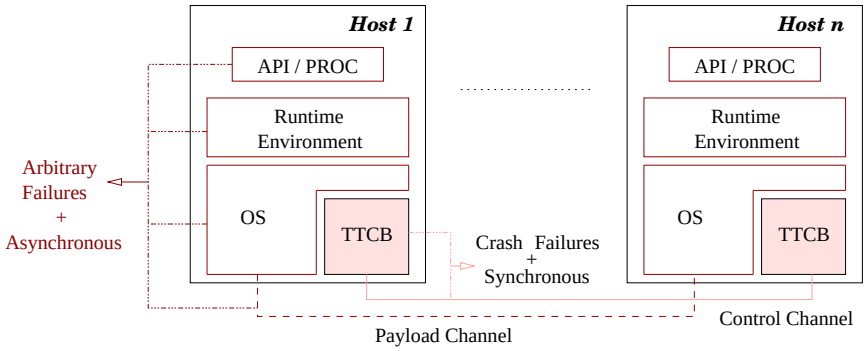


Fig. 1.1. Interconnection of the *TTCB* modules within a payload network

Each host is assumed to have a local module called the *local TTCB*. These modules are also assumed to be interconnected by a completely secure *Control Channel* and form what is called the *Distributed Trusted Component* (*DTC*). A *TTCB* assists the applications running between participants in the concerned hosts which are interconnected by a vulnerable *Payload Channel* forming a payload system subject to arbitrary byzantine failures.

The fail-controlled with distributed trusted components intrusion-tolerance strategy which has been proposed by the *MAFTIA* team states that some

global actions can be trusted despite a generally malicious communication [14]. The *TTCB* allows the payload system to take advantage of its possible synchronism by assisting the application to determine useful facts about time (such as the execution time of an operation, its duration, etc). However, this is trusted to the following extent: It is assumed to be not feasible to subvert a *TTCB*; but it may be possible to interfere with its interaction with software components.

We have chosen to rely on tamperproof *TTCB* modules in the design of our protocol because a timed behavior can be supported globally in an intrusion-resilient way. This is able to help limiting the potential damage of malicious behavior while having accurate information about the available QoS.

1.3.2 The QITAR protocol description

QITAR Assumptions

We assume available an ad-hoc network with medium density and bi-directional wireless links. We also adopt a reactive approach. The number of *TTCB* modules is set according to the hostility of the environment and the node density. Besides, we assume that only some mobile nodes can host such secure real-time components. We also decide to rely on these components to provide intrusion tolerance and secure QoS estimation. Furthermore, we assume that a mobile *TTCB*-equipped node is not allowed to leave the network and is already known by all the network nodes. Moreover, the traffic is session-oriented, where each unidirectional session is called a *flow*. Finally, we suppose the existence of a *Certification Authority* which delivers asymmetric key pairs as assumed in [18] and [19] along with *DHCP* server which is in charge of assigning a unique IP address to each *MN*.

Network model

The ad-hoc network will be divided into routing-zones, each of them will be managed by a *TTCB*-hosting node. This manager will be the direct neighbor of the managed *MNs* and a direct neighbor of other managers so that the routing-zones overlap. We denote the source node by S, the destination node by D, the *TTCB* module of the node managing the source by $S\_\ TTCB$, the destination' *TTCB* manager by $D\_\ TTCB$ and the intermediate *TTCB*s by $I\_\ TTCB$ as illustrated below in Figure 1.2. Each *TTCB* communicates locally with its agents (denoted by the letter *A)* in the same figure. Five phases can be considered: the neighborhood discovery phase; the mobile nodes registration phase; the route establishment phase; the maintenance phase; and the data transfer phase.

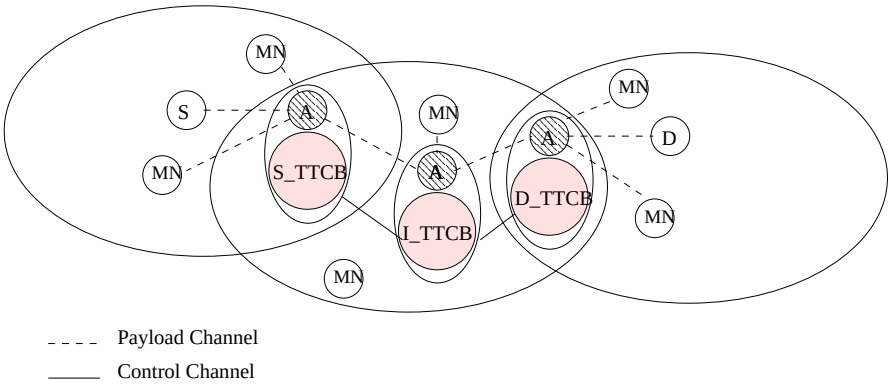


Fig. 1.2. Notations

TTCB*s' neighborhood discovery phase*

The managers have to enter in agreement in order to securely forward the signaling messages hop by hop on the control channel. Consequently, they must always have an accurate list of their $TTCB$ neighbors. $TTCB$ modules should periodically enter in agreement in order to update their neighborhood information. The period duration can be stated by the network manager. However, malicious agents may try to cause a denial of service by changing the agreement identifiers or refusing to propose a value for the agreement on time. Thanks to the $TTCB$ properties, the other neighbors will decide their values as if the attacked manager could not hear them, even though they cannot have a complete vision of the network topology before the next period.

Registration phase

Each mobile node has to register to a manager in order to become reachable and emit route requests. To secure the pending communication and the further ones, each MN should share a key with its S_TTCB. We assume that the registration phase is periodically executed in order to renew the symmetric key and change the manager in case of mobility. The registration procedure includes some mechanisms in order to tolerate the potential denial of service attacks.

Route establishment phase

The route establishment phase is divided into two sub-phases. The first securely establishes a path on the control channel in order to counter the wormhole attack and estimates the control information transmission delays so that

the reception of the data acknowledgments can be predicted. The second subphase tries to secure the estimation of the QoS provided on the payload channel since all the data will be transmitted on it.

A mobile node S that wants to establish a communication with a destination D originates a route request and asks the manager of the routing-zone it belongs to for a route while indicating its QoS requirements in terms of maximum delay and minimum bandwidth. S_TTCB will estimate the payload edge delay between it and the source then it will send a control route request. This request is processed by the intermediate $TTCBs$, say I_TTCBs, travels on the control channel, and contains the global timestamps values. If the QoS can be met, a control path is established on the same channel. The I_TTCBs are configured to counter DoS attacks and verify the integrity of the route request.

To have an accurate value of the effectively providable QoS while tolerating intrusions, we need to estimate it on the payload path corresponding to the pre-established control route. For this, S_TTCB creates a new agent then asks it to begin estimating the delay on the payload channel providing it with the list of the intermediate $TTCBs$ and the timestamp of the second subphase's beginning time. A second copy of the same payload route request will travel on the control channel so that the I_TTCBs can verify whether the packet is authentic and whether it has been sent by the predecessor on the pre-established route. When this request reaches D_TTCB, one can be sure that the required QoS can be met on the entire payload path. If the checked estimated bandwidth is correct, it will be reserved immediately, e.g during the request phase, otherwise, the agents must re-estimate the QoS and the $TTCBs$ must verify it during the reply phase; thus causing an important and unnecessary overhead. Because not all the nodes of the established route on the control channel are surely able to provide the required QoS, the network must release the reserved resources and update the throughput information as soon as possible, if the route is not used.

Maintenance phase

The managers' mobility breaks down the established routes so that the aforementioned phases are re-executed, without being sure that suitable paths can be found at the end. For this, we propose a rescue tentative which relies on the direct common neighbors of the affected managers in order to replace only the two broken links; thus reducing the overhead.

Data transfer phase

The data transfer phase takes place once a route on the payload channel respecting the QoS requirements has been established. Multiple routes can be found and may be useful when one or more I_TTCB leaves an active route. However, resources may still be reserved unnecessarily. Therefore, source S should send its data on only one route while the other path(s) should be

released shortly after. All the data must be acknowledged and securely transmitted.

1.4 QITAR design

To discover its neighborhood, each *TTCB* should periodically send a hello message to all existing *TTCBs* since all the modules are globally synchronized. Only the *TTCB*s which can "*hear*" the hello are able to decide a value. When a new *MN* enters the ad-hoc network, it should generate a registration request. Each *TTCB* hearing the request should send back a signed and encrypted response containing a shared key value that can be used to authenticate the initiator and encrypt the further exchanged messages. The concerned *MN* replies by a confirmation message in order to be registered. However, malicious mobile nodes and attacking agents may try a denial of service attack by generating many registration requests with unknown addresses, initiating many registration requests within a short period of time, or altering the needed control packets. To tolerate such intrusions, each *MN* must be authenticated. Moreover, each manager must be configured to process a limited number of registrations and frequently renew its agents.

A *MN* wishing to communicate with a destination should generate a route request containing its QoS requirements. *S_ TTCB* receiving it starts by estimating the edge payload delay between it and the source node as detailed in section 1.5. If the delay is inferior to the specified one, *S_ TTCB* appends its identity and the time of transmission to a control route request. Then it broadcasts it to its *TTCB* neighbors through the control channel. The *I_ TTCBs* process the request by adding their identifiers and the reception's timestamp and create a new table entry for this flow if the delay requirements can be fulfilled. A control route request initiated by a source and reaching the same destination must be transmitted during a certain threshold interval only once. On the receipt of request, *D_ TTCB* estimates the delay on the final direct payload link and forwards the control route reply on the reverse path if the QoS can be met.

If an *I_ TTCB* does not receive the control route reply within a threshold period of time, the flow's table entry must be deleted; otherwise, it is confirmed. On receiving the replay, *S_ TTCB* can exactly determine each intermediate manager on the path and the corresponding delays on each control link. However, a malicious mobile node may try denial of service attacks. Therefore, each *TTCB* must be configured to reject the redundant messages. In addition, a malicious agent may not immediately forward the request or the reply messages in order to make the delay look higher than it really is. Consequently, *S_ TTCB* and *D_ TTCB* modules should frequently renew their agents and restart the edge delay estimation procedure if they do not receive the correspondent messages after a timeout.

To guarantee more robustness during the delays estimation on the payload pre-established path, we propose that the agent, say *Agent(n)*, at the nth *I_ TTCB,* denoted by *I_ TTCB(n)*, gives the payload route request to its local *I_ TTCB,* which adds the timestamp of reception, signs the control packet, gives it back to *Agent(n), and* forwards a non-signed copy on the control channel to *I_ TTCB(n+1)* if the QoS can be met. When receiving the signed request, *Agent(n)* sends it on the payload channel to *Agent (n+1)*. Since the control channel is faster than the payload channel, the copy sent on the control channel may arrive first so that *I_ TTCB(n+1)* may have to wait for the payload copy and compare the two received packets. If the copies are identical, *I_ TTCB(n+1)* asks for the available bandwidth and checks it. Then, it provides its *Agent(n+1)* with a new encrypted request message if the QoS can be met. If the pretended bandwidth value is not correct, *I_ TTCB(n+1)* should kill its agent and ask for a new bandwidth value. Moreover, if the two copies of the same route request are not identical, *I_ TTCB(n+1)* kills its agent and asks *I_ TTCB(n)* to manage a new agent by sending a kill request on the control channel. *I_ TTCB(n)* provides the newly created agent with a signed kill reply then forwards a non signed copy to *I_ TTCB(n+1)* which has to wait for the second signed copy. However, this procedure cannot be executed more than a fixed number of times. When the payload request message reaches *D_ TTCB*, this latter should estimate the QoS on the final edge link. If the QoS is met on the entire payload path, *D_ TTCB* should send back a reply message in order to inform *S_ TTCB* that the route is ready for data transmission.

The reservation management is performed as follows: *I_ TTCB(n)* assigns a token bucket to each flow if the required bandwidth is inferior to the available one. If the QoS cannot be met at the manager *(n+i)*, *I_ TTCB(n+i)* should send a release request to the intermediate *TTCB* neighbor on the reverse path in order to release the previously reserved resources.

To improve the performance of the maintenance phase, we propose that before quitting a route, *I_ TTCB(n)* asks *I_ TTCB(n-1)* and *I_ TTCB(n+1)* for the list of their *TTCB* neighbors in order to search for a common one. A common manager can initiate a QoS estimation on the payload channel and send a rescue reply message to inform *I_ TTCB(n-1)* and *I_ TTCB(n+1)* about the path modification if the requirements can be met. If there is not any common manager or if the common neighbors can not provide the required QoS, *I_ TTCB(n)* should send an error notification on the control channel in order to release the reserved resources on both direction. Every *TTCB* should frequently renew its agents and the rescue procedure will be retried for a configured number of times for further security.

The data transfer phase begins when *S_ TTCB* sends a data request message to the source node informing it that a suitable route has been found. A source node can receive many route replies since it can be managed by several *TTCBs*. The resulting multiple routes are useful when *I_ TTCBs* are likely to leave established active routes. However, managing multiple routes for the

same flow keeps resources reserved unnecessarily in the other paths. To address this issue, S should send its data only on one route and D should inform its D_TTCBs about any duplication using a signed release request. After a threshold period of time, if there are no received data packets, D_TTCB is allowed to release the route. However, S and D may be malicious and cause denial of service attacks by unnecessary requesting routes or by pretending duplications. Consequently, if there are no data received after a timeout, the concerned S_TTCB should send a release message on the control channel and stop processing other route requests issued from the same source during a threshold delay. In addition, S_TTCB must forward the received request release signed by D to S. S must collect all these requests, and if all the established routes have been released on demand, it will send back the control packets to all its managers so that they can deduce that the destination node is malicious.

To implement the previously described phases in a secure manner, our protocol uses several signaling messages mainly transmitted on the control channel. A non exhaustive list of messages is depicted as follows:

- A ***NeighReq*** request is periodically sent in order to determine the $TTCB$ neighborhood.
- The mobile node registration needs the exchange of a ***RegReq*** request, a shared key ***SK*** message, and its confirmation reply ***SKConf***.
- A mobile node will initiate the route establishment phase by indicating its QoS requirements in a ***RReq*** request.
- The beginning of the edge delay estimation is marked by a ***DelayReq*** and the correspondent reply reception ***DelayRep*** will permit the delay calculation.
- A ***CRReq*** will travel on the control channel in order to establish a control path while the arrival of the correspondent ***CRRep*** confirms that route.
- The QoS estimation on the payload channel begins by the transmission of a ***PRReq*** request and ends by the reception of the correspondent ***PRRep***.

- Two messages, ***KReq*** and ***KRep,*** may be used for killing the malicious agents.
- The ***Release*** message orders the release of a route.
- The maintenance phase employs the ***ResNeighReq***, ***ResNeighRep***, ***ResReq*** and ***ResRep*** messages in order to ask for rescuing neighbors and check whether the required QoS can be met.
- A ***Bye*** message is sent by the leaving manager and an ***ErrrorNot*** is forwarded to annunciate the breakage of the established route and order the release of the reserved resources.
- A ***DataReq*** invites the MN to transmit its data while a ***NAck*** indicates that there is no suitable route found.

1.5 Secure bandwidth and delay estimation

Delay estimation

We provide in this section a secured procedure for the estimation of delays using the $TTCB$ time-related services. Control channel delays will be used to determine the reception time of data acknowledgment and detect malicious agents.

To estimate the communication delay on the edge payload link, a managed MN should first initiate a route request. The manager's agent which receives the control packet informs S_TTCB which triggers the duration measurement service and order a delay request message. The local agent must then relay the message to the source node then wait for a reply in order to relay it back to S_TTCB. Once the correspondent reply is received, S_TTCB ends the service and provides an accurate result reflecting the edge delay. This procedure is summarized by Figure 1.3. For the sake of clarity, we assume that the delay of local communication duration between the agent and its $TTCB$ can be neglected.

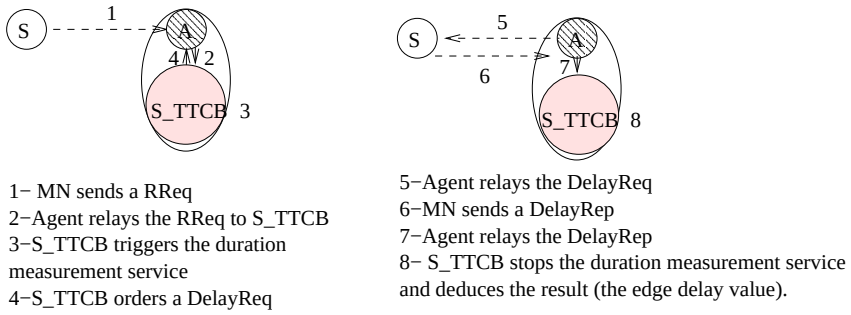


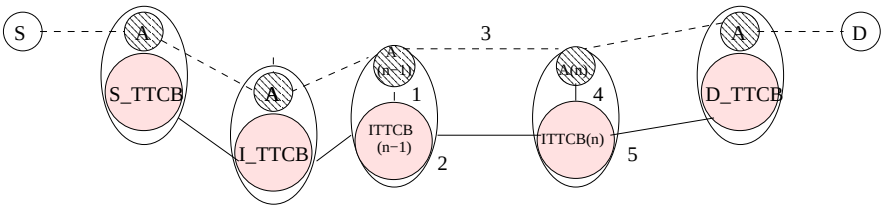
Fig. 1.3. Edge delay estimation

To estimate the delay on the control channel, S_TTCB should broadcast to its $TTCBs$ direct neighbors a control route request containing the pre-estimated edge delay $D_{S \rightarrow STTCB}$ and the current timestamp. When receiving the request, each $TTCB$ reads the global timestamp value then subtracts the timestamp indicated by S_TTCB from it in order to deduce $d_{STTCB \rightarrow TTCB}$(where $d_{X \rightarrow Y}$designates the delay between X and Y on the control channel). If the sum of both delays is inferior to the maximum delay allowed (e.g., if $D_{S \rightarrow STTCB} + d_{STTCB \rightarrow TTCB} < max - delay - allowed$), I_TTCB broadcasts the request further to all its $TTCBs$ neighbors, and so on. When D_TTCB receives the control packet, it sums up all the delays on the previous control links and computes a delay estimation. If the cumulated delay is smaller than the delay allowed, say:

$$D_{S\to STTCB} + \sum_{j=STTCB}^{ITTCB(f)} d_{j\to j+1} + D_{DTTCB\to D} < max-delay-allowed \quad (1.1)$$

D_ TTCB will send back a control reply to inform that a control path has been established.

The delay estimation on the payload channel is done as follows: *S_ TTCB* signs a payload request then gives it to its local agent, which is in charge of sending it to the next agent on the pre-established path. When *I_ TTCB(n)* receives that request from its agent, it begins by reading the global timestamp value and subtracting the timestamp value indicated by *I_ TTCB(n-1)* from it. The deduction of delay $D_{n-1\to n}$ on payload link $n-1 \to n$ is described by Figure 1.4.



1− Agent (n−1) gives the payload route request to TTCB(n−1)
2− TTCB(n−1)verifies if the delay requirements can be met. If it is the case, it adds its identifier and its timestamp value then gives a signed updated copy back to its agent and forwards a non signed one to TTCB(n)
3− Agent (n−1) forwards the route request to Agent (n)
4− Agent (n) gives the sigend copy of the route request to TTCB (n)
5− TTCB(n) consults the global timestamp value and subtracts the timestamp value indicated by TTCB(n−1) in order to deduce the payload delay on the link n−1 −−> n.

Fig. 1.4. Delay estimation on the payload channel

Finally, *I_ TTCB(n)* sums up the delays on the previous links forming the path (i.e., computes $D_{S\to STTCB} + \sum_{j=1}^{n-2} D_{j\to j+1}$, where *TTCB(1) = S_ TTCB*) and adds its own delay. The delay on the payload channel is then given by:

$$D_p = D_{S\to STTCB} + \sum_{j=1}^{n-2} D_{j\to j+1} + D_{n-1\to n} \quad (1.2)$$

If D_p is inferior to the allowed maximum delay, *I_ TTCB(n)* adds its identifier and its timestamp value to the request then gives a signed copy to its local agent and forwards a control copy further. The same procedure is repeated and the payload request will reach *D_ TTCB* only if the QoS can be met on the payload path. It is worth noticing that the computed results include the transmission and processing delays.

Bandwidth estimation

In ad-hoc networks, the resource estimation is often based on the statistical information provided by the *MAC* layer. In fact, the *MNs* monitor the channel status to determine the busy and idle periods of the shared wireless media then deduce the available bandwidth. This means that the available bandwidth on the payload channel is computed by insecure agents which may provide false estimation. To secure the bandwidth estimation, the provided values need to be verified using the delay information given by the *TTCBs*. Our secured bandwidth estimation on the payload channel at *I_ TTCB(n)* begins when *Agent(n-1)* sends the payload route request to *Agent(n)* in order to verify if the payload link $n-1 \rightarrow n$ is suitable for the correspondent flow. As *I_ TTCB(n)* determines the payload delay information by consulting its timestamp value and the timestamp value provided by *I_ TTCB(n-1)*, it should subtract the processing time proportionally to the payload route request length in order to get the transmission delay $D_{n-1 \rightarrow n}$, it can then deduce the throughput on the link $n-1 \rightarrow n$ using the formula [16, 13, 17]:

$$Throughput_{on\,the\,link} = \frac{Packet\,Size}{D_{n-1 \rightarrow n}} \tag{1.3}$$

Moreover, to detect selfish nodes, we assume that the availability of the wireless channel can not be less than a threshold value reflecting the minimal QoS level that has to be offered by the network. As *Agent(n)* provides its local *I_ TTCB(n)* with a computed bandwidth information, *I_ TTCB(n)* will be able to verify the accuracy of the presented value by evaluating the channel availability ratio using the formula :

$$BW_{Available} = (1-u) * Throughput_{on\,the\,link} \tag{1.4}$$

[16, 13, 17] where $1-u = \frac{Idle\,times\,in\,window}{window\,duration}$ is the link availability and $idle\,times\,in\,window$ represent the fraction of time within which the agent is sensing the channel as being idle (e.g., the channel is not idle if the agent is transmitting or receiving packets or if some other nodes within its neighborhood are transmitting data or RTS/CTS packets).

In fact, let us assume that the estimated throughput equals y and that *Agent(n)* pretends an available bandwidth value equaling $x\,Mb/s$. Therefore,

$$BW_{Available} = (1-u) * Throughput_{on\,link\,n-1 \to n} = (1-u) * y = x\,MB/s$$

Thus, $(1-u) = x/y$. If $1-u \leq 1$ and $1-u \geq minimal\,threshold\,value$, we can assume that the agent is neither attempting a denial of service attack nor trying to avoid routing packets; thus, the probability that it has pretended a correct bandwidth value is high.

To calculate the available bandwidth, *Agent(n)* should first calculate the channel utilization ratio R. Suppose the last channel utilization ratio is $R_{t-1}$and the channel utilization ratio measured in the current sampling time window is $R = \frac{channel-busy-period}{window\,duration}$. Then, the current channel utilization ratio is given as $R_t = \alpha R_{t-1} + (1-\alpha)R$ where α is a smoothing constant $\alpha \in [0,1]$. The agent can deduce its available bandwidth at time t using the formula

$$BW_t = W(1-R_t) \tag{1.5}$$

where W is the raw channel bandwidth (*e.g.*, $2\,Mb/s$ for a standard 802.11 radio) [12].

1.6 QITAR security features

To enhance the security of the routing, the ***QITAR*** protocol tried to prevent the denial of service attacks. In fact, malicious mobile nodes may collude or independently overflow the managers by multiple requests, unnecessarily ask for the routing service or replay the same messages. Moreover, the malicious agents may corrupt the control packets or refuse to forward them to the local *TTCB* modules. The impersonation attack may also be a form of denial of service. In fact, if the *MN*'s identity has been spoofed, the *MN* can no longer register to the mangers. In addition, a modified request cannot reflect the required QoS and the victim node will not be served. To address these issues, we have modified the *TTCB* kernel developed in *MAFTIA* project [14] in order to configure threshold values related to the number of processed requests, and the number of overall processed messages. Finally a procedure to stop providing routing services is added in the case of malicious source and destination. A threshold value related to the number of re-attempts is also added.

Moreover, an asymmetric cryptography scheme is used in order to authenticate each new-arriving mobile node, it will then be replaced by a symmetric one when the *MN* shares a key with its managing *TTCB*; thus reducing the processing efforts, accelerating the communications and guaranteeing the authenticity of the sender and both the integrity and confidentiality of the exchanged messages on the payload edge link. This shared key is periodically renewed in order to guarantee more robustness and cope with the possible mobility of the managing-nodes.

A man in the middle may sniff many registration requests coming from *MN*s that belong to different routing zones then redirect them to a distant

TTCB-hosting node that is unable to manage them. The victim manager will waste its resources in processing the request and sending back the shared key message. The hacker then relays the control message and its confirmation and the manager finally registers an unreachable *MN*. To counter this attack, the *MN* authentication can use the combination of MAC and IP addresses.

A major advantage of using the *TTCB* kernels is the protection against the wormhole attacks. In fact, after the secure delay estimation on the reliable control channel, a malicious agent on the path between the source and the destination will not be able to tunnel the route request message to another one during the delay estimation on the payload channel. In fact, the intermediate *I_ TTCBs* already know all the nodes forming the route. Moreover, they can easily verify that the control message has been forwarded by the previous *I_ TTCB* thanks to the **PRReq** copy traveling on the control channel. Let us demonstrate this assertion by the example below:

- **First case:** *Agent(n)* and *Agent(m)* are two colluding ends of a wormhole tunnel and are both malicious; however, *Agent(m)* does not belong to the route. *Agent(m)* will forward the **PRReq** packet to a well-behaving *Agent(l)* belonging to the route. *I_ TTCB(l)* will discover that the received **PRReq** was not forwarded by its predecessor on the route because the message is not signed by the predecessor. Consequently, it ignores it.
- **Second case:** *Agent(n)* and *Agent(m)* are two colluding ends of a wormhole tunnel. They are both malicious and belong to the route. All the *I_ TTCBs* including between *I_ TTCB(n+1)* and *I_ TTCB(m-1)* will not receive the payload request copy at time. Moreover, *I_ TTCB(m)* will receive a payload copy that is not signed by its predecessor on the route. As a result, the malicious agents will be killed and the delay can then be correctly estimated.

Finally, because it is assumed that a *TTCB* can not be compromised, it is possible for a group of *TTCBs* to support a reliable protocol with $f + 2$ replicas, requiring an attacker to compromise $f + 1$managers in order to cause an intrusion (the traditional approach requires $3f + 1$ replicas to tolerate f failures) [14]. Since we assumed that the mobile *TTCB*-equipped nodes are not authorized to leave the network and that they are already known by all members, it is possible to determine the number of such managing nodes depending of the hostility degree of the environment.

We believe that our protocol is more efficient than the others because it has addressed especially dangerous attacks such as the denial of service and the wormhole while trying to provide high QoS guarantees.

1.7 Conclusion

The focus of this paper is on providing a *TTCB*-based routing protocol which guarantees QoS constraints such as delays and bandwidth to independent

flows, while tolerating some common attacks. Our protocol implements cryptographic operations in order to guarantee the integrity and the confidentiality of the exchanged data. In addition, it secures the QoS estimation by verifying the pretended bandwidth values using the timestamp data provided by the *TTCB*s. We have provided a modified version of the *TTCB* kernel developed by MAFTIA to support new functionalities such as the determination of the *I_ TTCBs* neighbors and the registration of the managed mobile nodes.

References

1. C. Perkins, E. M. Royer and S. R. Das, "Ad hoc On-demand Distance Vector routing", In IETF Internet Draft, draft-ietf-manet-AODV-12.txt, November 2002.
2. D.B Johnson, D.A Maltz and J. Broch, "DSR: The dynamic Source Routing Protocol for Multi-hop wireless Ad Hoc Networks", in Ad Hoc Networking, ch. 5, pp. 139-172, Addison-Wesley, 2001.
3. C.E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance Vector Routing (DSDV) for Mobile Computers", in ACM SIGCOMM'94, October 1994.
4. N. Ferreira and P. Verrissimo, "Complete Specification of APIs and Protocols for the MAFTIA Middleware", MAFTIA deliverable D9, 2002.
5. S. Bouam and J. Ben Othman, "Securing Data Transmission and Retransmissions Management in Ad Hoc Networks", Proceedings of the International Conference on Wireless Networks, ICWN '04, June 2004.
6. M. G. Zapata and N. Asokan, "Securing Ad Hoc Routing Protocols", in WiSe'02, September 2002 .
7. P. Papadimitratos and Z. J. Haas, "Secure Routing for Mobile Ad Hoc Networks", In proceedings of the CNDS conference, San Antonio, TX, 2002.
8. B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru and H. Rubens, "ODSBR: An On-Demand Secure Byzantine Routing Protocol", Technical report, October 2003.
9. C. E. Perkins, E. M. Royer and S. R. Das, "Quality of Service for Ad hoc On-demand Distance Vector Routing", IETF Internet Draft, work in progress, July 2000.
10. G.S. Ahn, A.T. Campbell, A.Veres and L.H. Sun, "SWAN: Service Differentiation in Stateless Wireless Ad Hoc Networks", Proceedings of IEEE INFOCOM 2002, June 2002.
11. D.Powell and R. Stroud, "Conceptual Model and Architecture of MAFTIA", MAFTIA deliverable D21, 2003.
12. K. Xu, K. Tang, R. Bagrodia, M. Gerla, M. Bereschinsky, "Adaptive Bandwidth Management and QoS Provisioning in Large Scale Ad Hoc Networks", IEEE MILCOM'03, Boston, Massachusetts, October 2003
13. S. Lohier, S. M. Senouci, Y. M. Ghamri. Doudane and G. Pujolle, "A reactive QoS Routing Protocol for Ad Hoc Networks", European Symposium on Ambient Intelligence (EUSAI2003), Eindhoven, Netherlands, November 2003, Lecture Notes in Computer Science, Springer Verlag.
14. D. Powell and R. Stroud, "Conceptual Model and Architecture of MAFTIA", MAFTIA deliverable D21, 2003.

15. Y. C. Hu, A. Perrig and D. B. Johnson, "Packet Leashes : A Defense against Wormhole Attacks in Wireless Ad Hoc Networks", Technical report TR01-384, Revised September 2002.
16. M. Kazantzidis and M. Gerla , "End-to-end versus Explicit Feedback Measurement in 802.11 Networks", In Seventh IEEE Symposium on Computers and Communications, 2002.
17. H. Badis, A. Munaretto, K. Al Agha and G. Pujolle, "QoS for Ad hoc Networking Based on Multiple Metrics : Bandwidth and Delay", IFIP/IEEE MWCN 2003, Singapore, October 27-29, 2003.
18. W. Yu, Y. Sun and K. J. R. Liu, "HADOF : Defense Against Routing Disruptions in Mobile Ad Hoc Networks", in Proc. IEEE INFOCOM'04, Hong Kong 2004.
19. B. Awerbuch, D. Holmer and H. Rubens, "Provably Secure Competitive Routing against Proactive Byzantine Adversaries via Reinforcement Learning", John Hopkings Univ, Tech. Rep, May 2003.