UDC 621.391.1

MULTIROUTE CROSS-LAYER TRANSPORT PROTOCOL WITH RELIABLE DATA DELIVERY

Iurii Iu. Voitenko

National Technical University of Ukraine "KPI", Kyiv, Ukraine

A multiroute cross-layer transport protocol with data delivery capabilities for mobile networks is proposed. Protocol combines routing functionality with reliable data delivery. It is shown that finding multiple routes to the destination can be successfully combined with a connection establishment for connection-oriented protocols such as TCP. Establishing a connection over multiple routes can significantly decrease the time of the connection recovery when some routes become unavailable thus increasing total throughput of transport protocol in mobile networks environment. Performance analysis of the proposed protocol indicated that in mobile sensor networks with 5-10% of packet losses its throughput is approximately 60% higher than throughput of TCP-Reno2 flowing over shortest path.

Introduction

The loss of data is unavoidable occasion in any communication system. In mobile sensor networks every node serves as a router relaying packets to the other nodes and at the same time it is free to move in any direction. Therefore such networks are even more prone to data losses than wired ones. According to OSI model reliable data delivery and congestion control are carried out by transport layer protocols. Standard TCP treats every packet drop as congestion in the network and decreases the data rate [1, 2]. In mobile sensor networks, packet drops are caused not only by congestion but primarily by changes in the network topology, which leads to unpredicted changes in the performance of available routes. This paper discusses various reasons of deliver data inability in mobile sensor networks and proposes a new transport protocol designed to operate in highly mobile environment.

Mobile sensor networks present many challenges, especially when real time application must be supported in terms of providing QoS guarantees. One of the biggest challenges is routing since all nodes can move. Variety of routing protocols has been developed. Some of them such as AODV and DSR are presented in RFC documents [3, 4]. All of them are designed as independent algorithms which can operate with any transport protocol in accordance with decomposition of OSI layers. However, in the case of severe performance degradation or route failure it can take significant time to find a new route while transport layer will continue to deliver data packets on a network layer. This can lead to unacceptable quality of service on application layer.

Another challenge for mobile sensor networks is congestion control on a transport layer. Such classical TCP mechanisms as congestion avoidance, fast retransmit and recovery mechanisms do not entirely correspond to the wireless environment which this protocol operates in, hence proper adaptation is needed. Different adjustments to TCP for operating in mobile sensor networks have been presented [5]. To overcome the problem of connectivity loss, some cross layer techniques are proposed [5]. This way allows collecting the information on radio links, delays and available bandwidth on each hop which can be used for dynamic adaptation to the network changes. This circumstance is important when applications (usually multimedia) have strict requirements on delay, bandwidth and jitter. To maintain the connectivity between end-users and guarantee QoS, the use of multiroute data delivery scheme is proposed. It allows redirecting the current traffic to bypass unstable relay nodes without spending additional time on routing. The data delivery scheme proposed can effectively prevent congestions in mobile sensor networks. Depending on QoS requirements requested by application process, the transport protocol can choose routes with appropriate parameters in terms of delay, throughput and reliability. In this case, sharing information across layers is required. More generally, network and transport layers in mobile sensor networks should be combined in one protocol and should have access to link and application layers. This way simplifies collecting information on routes and link performance and adjustment to changes in mobile sensor network. In this paper, we show that the use of cross layer design combined with multiroute data delivery can significantly increase total throughput and stability of end-to-end transport connection. The paper contains an overview on establishing a connection between TCP and its adjustments for mobile sensor networks, a design of a new cross-layer transport protocol, as well as a simple model for data transmission in mobile sensor networks and performance analysis of the presented protocol.

Connection establishment in mobile networks

All connection-oriented protocols require a "handshake" before they can exchange application data. This phase is important to track the presence of the respective service on the remote host. TCP uses three-way handshake prior to sending application data. During this phase sender and receiver notify each other on buffer size and maximum segment size (MSS) which can be sent to the network layer. In mobile sensor networks when using reactive routing protocols, establishment of a connection is possible only after routing phase. For instance, in AODV routing protocol when sender desires to send a message to some destination node and does not already have a valid route to that destination, it initiates a path discovery process to locate the other node. It broadcasts a route request (RREQ) packet to its neighbors which then forward the request to their neighbors, and so on, until either the destination or an intermediate node with a "fresh enough" route to the destination is located. Once route request reaches the destination or an intermediate node with a fresh enough route, it responds by unicasting a route reply packet (RREP) back to the neighbor from which the route request was first received. When the route reply packet reaches the sender, TCP sends first connection establishment packet (SYN packet) to the remote host. Actually, TCP sends packet before routing but it will remain in transmit FIFO buffer until route is found. It should be noted that the standard TCP has no means of explicit route assignment and "knows" nothing on which exactly route is found.

Another important issue is data delivery over multiple routes. Assume that another routing protocol capable to discover multiple routes from source node to the destination node is used. In this case, a route pool can be used to maximize data flow in the network. However, neither standard TCP nor its adjustments have no multihoming features which can benefit from multiple routes. The use of SCTP transport protocol having multihoming capability inbuilt is proposed in [6]. However, SCTP refers to a situation where a node can choose between several paths to reach a destination either by having multiple interfaces to choose from or by the network that is connected to the Internet by several routers or by routers with several interfaces. In other words, SCTP operates with multiple interfaces rather than routes.

According to TCP specification as soon as first SYN packet reaches destination node, server side socket transits to a "half-open" state, allocates respective buffers and structures and sends back SYN ACK packet. If more SYN packets with the same sequence number are already reached, allocated socket connection will reset [1]. This is fundamental issue which restricts standard TCP from using broadcasting techniques. However, in order to decrease the time of establishing a connection, route discovery process can be combined with sending SYN packets. This process is illustrated in Fig. 1, where source node A tries to establish a connection with destination node H broadcasting SYN packets. Even though network layer specifies source and destination addresses (A and H), broadcasting can be done on a link layer.

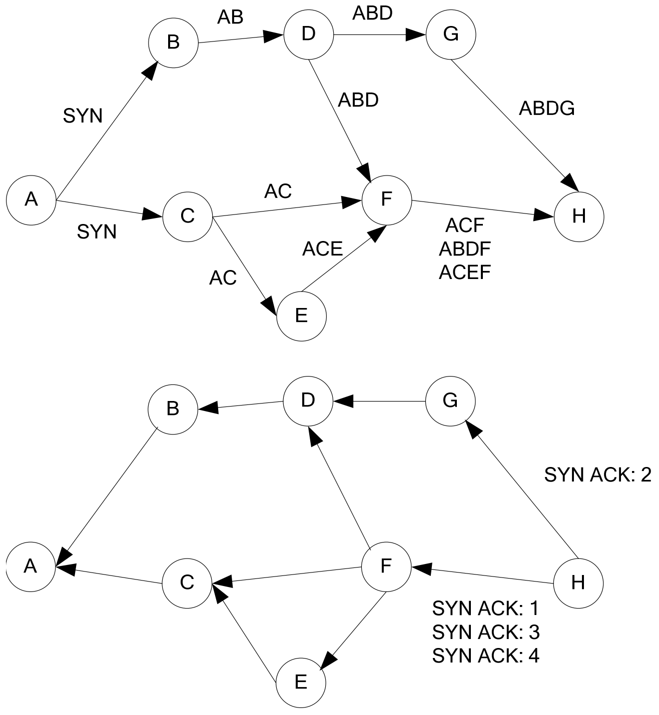

Fig. 1. Establishing of a multiroute connection.

In reactive routing protocols, such as AODV, DSR, TORA, route discovery process employs broadcasting of route request packets. Therefore, if intermediate nodes will propagate such packets even though they might have a "fresh enough" route, destination node can expect several copies of the same packet delivered over multiple routes. To provide additional multiplexing capabilities for the remote host, each packet must also contain unique session ID. For example, if IPv6 is used as underlying network protocol, it can be 20-bit flow label.

According to example shown in Fig. 1 (bottom), the destination node H is able to distinguish a route of each incoming SYN packet and replies it by unicasting and marking SYN ACK with session and route IDs. Thus destination node H after allocating socket does not discard every incoming SYN packet but simply assigns its route an ID, for instance, by incrementing a counter of incoming packets and appends it to the session routing table.

As first SYN ACK packet with proper session ID is received by source node (usually via shortest path), it extracts transport layer information such as receiver congestion window size W_r, MSS of the route, *route ID*, initializes session routing table and immediately starts sending available data packets over this route. These packets also serve as probes to determine reliability, throughput and delay of the discovered route.

Suppose that sending ACK packet according to the “three-way handshake” is done over the same route as SYN ACK. So, source node as a client and destination node as a server have established a connection over multiple routes using broadcasted SYN packets to avoid routing phase and for saving time. But once it is done, other fundamental questions appear immediately: which route is better and which route is more reliable and stable. Cross layer protocol design is an excellent candidate to address all those questions.

Cross-layer protocol design

In this section we discuss the implementation of the transport protocol in order to benefit from multiple routes in mobile sensor networks. As it was noted before, as soon as first SYN ACK reaches source, the node initializes session routing table. This table contains statistical information on available routes collected in real time. This information includes data about number of hops, MSS, number of sent and dropped packets as well as RTT in milliseconds. If new SYN ACK packet arrives, its routing information will be appended to this table. Remote host, i.e. destination node or server, has the same session routing table, except for sent packets in the case if no packets are lost. Table 1 shows session routing table on client side after processing 4 SYN ACK packets and 1 data ACK packet for the case when MSS is equal to 1460. In this table, the sign RTT refers to data stated as an example.

Table 1. Session routing table on client side.

N	Route record	Hops	Route ID	Sent	Lost	RTT
1	ACFH	3	1	2	0	4
2	ABDGH	4	2	1	0	6
3	ABDFH	4	3	1	0	10
4	ACEFH	4	4	1	0	12

Both client and server can assess stability of each route by calculation loss percent λ per route taking into account total number of sent packets:

$$\lambda_i = LOST_i \,/\, SENT_i \tag{1}$$

Retransmission timeout

Packets lost in the network should be recovered through its retransmission. Transport protocols use different mechanisms to notify the source about lost packet, so that it can retransmit them. These could be either positive acknowledgement (ACK) packets which notify sender on successful data packet delivery with particular sequence number or negative acknowledgement (NACK) packets which explicitly notify on data packet losses. In both cases, sender detects and retransmits the lost data packets. To ensure reliability of connection, sender also sets a retransmission timeout for every data packet. Its value is based on simple statistical information such as round-trip time (RTT). According to fast retransmission mechanism employed in latest versions of TCP reception of 3 duplicate, ACK packets trigger immediate retransmission of lost segment. However if sender did not send enough packets to generate 3 duplicate ACK ones, then unacknowledged data are retransmitted on timeout. To calculate appropriate retransmission timeout following formulas are used [2]:

$$SRTT = (1-\alpha)\cdot SRTT_{OLD} + (\alpha \cdot RTT_{LAST}), \tag{2}$$

where *SRTT* is “smoothed” round-trip time; $0 < \alpha < 1$. Usually $\alpha = 1/8$, then (2) can be expressed as:

$$SRTT = \frac{7}{8}\cdot SRTT_{OLD} + \frac{1}{8}\cdot RTT_{LAST}. \tag{3}$$

In RFC 1122 [11], the following retransmission timeout (RTO) calculation is recommended:

$$RTO = SRTT + 2\cdot SDEV, \tag{4}$$

where *SDEV* is “smoothed” deviation, to calculate which the absolute deviation *DEV* is found:

$$DEV = | RTT_{LAST} - SRTT_{OLD} |. \tag{5}$$

Then SDEV can be calculated as:

$$SDEV = \frac{3}{4}\cdot SDEV_{OLD} + \frac{1}{4}\cdot DEV \tag{6}$$

Formulas (2)—(6) do not support multiroute capabilities since they imply the calculation of one timeout value regardless of routes number available. Accordingly, since session routing table contains several RTT, it is reasonable to calculate its own RTO for every discovered route using formulas (2)—(6).

Yet another important issue of transport layer is efficient congestion control and connection maintenance. In mobile sensor networks, resolving this issue is much more complicated due to inability to define a real cause of packet loss: congestion, topology change, wrong checksum or other reason.

Using cross layer information, the exchange may help to define some reasons such as low signal-to-noise ratio on the radio link or wrong checksum on network layer but not all of them. Moreover, in mobile sensor network there is a multihop environment where the packet can be discarded at any intermediate node with no ability to predict it. Some protocols propose to use link layer ACK in order to notify about topology change. However, this may lead to unacceptable QoS at application layer.

Ignoring the initial slow-start period when a connection begins and assuming that losses are indicated by triple duplicate ACK or timeouts, TCP congestion control consists of linear (additive) increase in congestion window (*cwnd*) of 1 *MSS* per RTT and then a halving (multiplicative decrease) of *cwnd* on triple duplicate ACK event or timeout. Whenever new ACK arrives data sender increases congestion window (*cwnd)* by (*MSS/cwnd*) bytes.

Assume that aforementioned logic but with multiple routes available within established transport connection is used. After first SYN ACK reception, client sends first data packet. By the time it receives ACK for data packet it has already 2 routes. An ACK server also notifies client about its congestion window size W_r and client increases its congestion window W_s by 1 *MSS* segment thus sending 2 data packets. Both client and server sides update their session routing tables and calculate loss percent λ for each available route either when new packet arrives or on timeout. Routes with lower λ are statistically more stable in terms of packet drops. As soon as client has 2 routes in its table, it sends 1 packet over this new route and the remaining packets over previously discovered route.

Thus, in our example, each available route bears a data packet. Since in mobile sensor networks the probability of out-of-order packets delivery is much higher than in fixed networks due to dynamically changing route performance, one can assume that no duplicate ACK are generated and server always replies with cumulative ACK, i.e. one ACK packet per group of data packets is sent over route with lowest λ.

To benefit from cumulative ACK, server must also maintain timeout interval *t* at which it accumulates data packets. This interval can be defined as:

$$t = W_s \cdot RTT_{AVG} \tag{7}$$

where W_s is the number of packets "in-flight"; RTT_{AVG} denotes average RTT for all routes. As soon as at least 1 data packet arrives, server determines W_s.

Consider flow scenario where 4 data packets over different routes and 4 ACK packets have been successfully delivered. By the time of last ACK arrival, the client has already 4 routes available. It increases congestion window W_s to 4 MSS and send 1 packet per route.

Consider another scenario with 5 data packets. Suppose that packets 3 and 5 are lost, which is more realistic scenario to happen in mobile sensor networks. Let as analyze it step by step. Since $W_s \le R$, where R is a number of available routes, client send data packets one per route and expects a cumulative ACK over the most reliable (stable) route with lowest λ. Server upon receiving first data packet determines total amount of packets "in-flight" W_s and calculates waiting period for other missing segments using (7).

After data accumulation period elapsed, server sends back ACK which indicates that packets 3 and 6 are missing. Server expects arrival of retransmitted packets within another awaiting period using formula (7). ACK packets are much smaller in size than data packets therefore the loss probability is also smaller and they reach destination faster.

If no data packets are received within another waiting period, server retransmits ACK over most "distant" route to the route used previously. To determine such route, consider session routing tables on server and client sides, represented in Tables 2 and 3, where RTT are stated as an example. Table 2 shows the server side statistic which occurs before sending second ACK packets.

Table 2. Session routing table on server side.

N	Route record	Hops	MSS	Route ID	Sent	Lost	RTT
1	ACFH	3	1460	1	3	1	10
2	ABDGH	4	1460	2	1	0	15
3	ABDFH	4	1460	3	1	0	30
4	ACEFH	4	1460	4	1	0	40

Table 3 shows client side statistics after arrival of second ACK ($\text{counter} = 2$).

The term "distance" means the modulo 2 sum between routes presented as vectors:

$$D_{ij} = R_i \oplus R_j,\ R_i, R_j \subseteq R \tag{8}$$

Table 3. Session routing table on client side.

N	Route record	Hops	MSS	Route ID	Sent	Lost	RTT
1	ACFH	3	1460	1	3	1	10
2	ABDGH	4	1460	2	3	0	15
3	ABDFH	4	1460	3	2	0	30
4	ACEFH	4	1460	4	2	1	40

Two routes are called as *independent* if the first route contains such nodes which are not present in the second route, otherwise routes are *dependent*. Table 4 shows distances between all available routes.

Table 4. Distance table.

N	Route record	1	2	3	4
1	ACFH	-	5	3	1
2	ABDGH	5	-	2	6
3	ABDFH	3	2	-	4
4	ACEFH	1	6	4	-

Formula (8) in relation to both communicating sides can be generalized as follows:

$$D = \max \bigcup_{i=1}^{k-1} R_{sent} \oplus R_i \ , \tag{9}$$

where R_{sent} is a set of routes (vector union) used in the last transmission round.

Thus, server chooses route ABDGH for the second ACK. Client waits for cumulative ACK within RTO period. Using formulas (2)—(7), it is easy to see that $t < RTO$, so client has enough time to wait for ACK before retransmission. If no ACK arrives, client halves its congestion window and chooses alternative routes according to (9), but in our example it receives second ACK. This ACK packet carries information about loss of 2 data packets (3 and 6) and previous ACK via incremented ACK counter. It also serves as implicit indication that shortest route ACFH (route ID=1) is currently unable to deliver packets (both data and ACK packets have been lost), therefore, a topology may change.

To determine failure nodes N_{fail} by client, the following matrix expression for routes can be obtained:

$$\begin{array}{ccccccccc} & A & B & C & D & E & F & G & H \\ 1. & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 2. & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 3. & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 4. & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{array} \tag{10}$$

$$N_{fail} = inv(\bigcup_i R_i^{ACK}) \bigcap (\bigcup_i R_i^{NACK}) \ , \tag{11}$$

where R^{ACK} defines a set of routes successfully delivering packets; R^{NACK} denotes a set of routes where packets were lost; inv specifies an inversion function (bitwise NOT).

Using formulas (9)—(11), we can obtain that failure nodes are C and E. Since node C is included only in failed routes 1 and 4, client should bypass this node. Then, client chooses routes 2 and 3 which do not contain node C. For the next transmission round, client sorts session routing table by λ and chooses most stable routes.

Performance analysis

In this section, we will evaluate a performance of the designed multiroute cross-layer transport protocol and compare it with the performance of standard TCP with congestion avoidance, fast retransmit and fast recovery mechanisms (version Reno-2) flowing over shortest path. Assume that application layer generates data for 15 transmission rounds in accordance with a network scheme presented in Fig. 1. Let each round contains MSS packets. We will denote rounds contained 1 data packet loss by mark *, as well as rounds with a data packet and ACK packet losses by mark **. At round 4 packet is dropped in node C, at round 8 in node F, at round 13 in nodes F and E. To calculate RTT parameters, initial values for each route were taken from Table 3. To solve the considered TCP problem, we use a fast retransmit algorithm at round 13. Table 5 shows timer parameters of TCP. All parameters are calculated by formulas (2)—(7) using initial value SDEV = 1500 ms.

Table 5. Timer parameters of TCP Reno-2.

Round	MSS	SRRT old, ms	RTT last, ms	SRTT new, ms	SDEV, ms	DEV, ms	RTO, ms
1	1	0.0	10.0	0.0	1500.0	10.0	3000.0
2	2	0.0	20.0	2.5	1127.5	20.0	2257.5
3	4	2.5	40.0	7.2	850.6	37.5	1708.4
4*	2	7.2	1708.0	219.8	647.3	1700.8	1514.5
5	3	219.8	30.0	196.1	910.7	189.8	2017.5
6	4	196.1	40.0	176.6	730.5	156.1	1637.5
7	5	176.6	50.0	160.7	586.9	126.6	1334.5
8*	6	160.7	1334.0	307.4	471.8	1173.3	1251.0
9	3	307.4	30.0	272.7	647.2	277.4	1567.0
10	4	272.7	40.0	243.6	554.7	232.7	1353.1
11	5	243.6	50.0	219.4	474.2	193.6	1167.9
12	6	219.4	60.0	199.5	404.1	159.4	1007.6
13*	7	199.5	30.0	178.3	342.9	169.5	864.1
14	4	178.3	40.0	161.0	299.6	138.3	760.1
15	5	161.0	50.0	147.1	259.2	111.0	665.6

Table 6 shows timer parameters of proposed multiroute cross layer transport protocol (MCTP). As can be seen, at round 4 TCP detects packet loss via timeout because there were not enough duplicate ACK packets to generate fast retransmit, so TCP had to retransmit missing segment on timeout. Another strategy has been used in the same situation by MCTP when cumulative ACK reached client immediately retransmitted missing segment over most "distant" route.

Table 6. Timer parameters of MCTP.

Round	MSS	SRTT old, ms	RTT last, ms	SRTT new, ms	SDEV, ms	DEV, ms	RTO, ms
1	1	0.0	10.0	0.0	1500.0	10.0	3010.0
2	2	0.0	25.0	3.1	1127.5	25.0	2280.0
3	4	3.1	95.0	14.6	851.9	91.9	1798.8
4*	2	14.6	45.0	18.4	661.9	30.4	1368.8
5	3	18.4	55.0	23.0	504.0	36.6	1063.0
6	4	23.0	95.0	32.0	387.2	72.0	869.3
7	5	32.0	110.0	41.7	308.4	78.0	726.7
8**	6	41.7	726.7	127.4	250.8	685.0	1228.3
9	3	127.4	45.0	117.1	359.3	82.4	763.7
10	4	117.1	95.0	114.3	290.1	22.1	675.2
11	5	114.3	110.0	113.8	223.1	4.3	556.2
12	6	113.8	125.0	115.2	168.4	11.2	461.8
13**	7	115.2	461.8	158.4	129.1	345.8	719.2
14	4	158.4	90.0	149.8	183.3	68.4	456.6
15	5	149.8	105.0	144.2	154.6	44.8	414.1

As follows from Table 6, at round 4 MSS value dropped to 2, so both protocols retransmitted 1 lost segment and 1 new data packet. The same arguments can be applied to the other loss events. Table 7 shows performance comparison for both protocols.

Table 7. Performance comparison.

Protocol	Data, bytes	ACK packets	Total delivery time, ms	Throughput, Kbps
TCP Reno-2	91500	64	3532	25.9
MCTP	91500	18	2192	41.7

Conclusion

In this paper, a cross-layer transport protocol with multiroute reliable data delivery capabilities is presented. The protocol proposed combines routing with transport layer features at connection establishing phase in order to decrease the connection time. It requires significant changes on server side behavior. This protocol should expect several copies of the same synchronization request and therefore should not reset connection. It should detect a route of every SYN packet arrived and reply by unicasting SYN ACK over detected route. This mechanism allows source node (client) to discover multiple routes to the destination node (server) within one transport connection.

After synchronization phase, both client and server sides collect connection statistics and put it in a session routing table. This table is updated every packet sending or timeout event. Having a set of routes simplifies maintenance of the connectivity between client and server and allows to shorten time to discover alternative path if loss is detected. In this paper, a method to define best alternative path for lost segments is presented. This method implies calculation of the most "distant" route among all available to the one used previously where "distance" is a number of nodes not included in previous transmission round.

A simple analytical model based on retransmission timeout and predefined loss events is presented. It allows easily compare the behavior of transport protocols and draw some conclusions on their performance in terms of number of data and ACK packets sent, total delivery time and throughput. Comparison between proposed multiroute cross-layer transport protocol and standard TCP Reno-2 run over shortest path has shown that in mobile sensor networks with 5—10% of packet losses total data delivery time in MCTP is approximately 60% higher due to significant changes in protocol retransmission logic and the use of multiple routes. One of the possible future works can be performance evaluation of proposed protocol to the other mobile sensor network transport protocols as well as its logic improvements.

References

1. Transmission control protocol, RFC 793 // Internet Engineering Task Force (IETF). — September 1981.

2. Allman M., Paxson V., Stevens W. TCP congestion control, RFC 2581 // Internet Engineering Task Force (IETF). — April 1999.

3. Perkins C., Belding-Royer E., Das S. Ad hoc ondemnad distance vector (AODV) routing, RFC 3561 // Internet Engineering Task Force (IETF). — July 2003.

4. Johnson D. The dynamic source routing protocol (DSR) for mobile ad hoc networks for IPv4, RFC 4728 // Internet Engineering Task Force (IETF). — February 2007.

5. Papanastasiou S., Ould-Khaoua M. Exploring performance of TCP Vegas in mobile ad hoc networks // International Journal of Communication Systems. — March 2004. — Vol. 17, Issue 2. — P. 163—177.

6. Steward R. Stream control transmission protocol, RFC 2960 // Internet Engineering Task Force (IETF). — October 2000.

Received in final form January 27, 2012