

# Experiences on enhancing data collection in large networks

INRIA Tech Report

Mohamed Karim SBAI

Projet Planète, INRIA Sophia Antipolis, France  
National School of Computer Science (ENSI), Tunisia  
Email: Mohamed\_Karim.Sbai@sophia.inria.fr

Chadi BARAKAT

Projet Planète, INRIA Sophia Antipolis, France  
Email: Chadi.Barakat@sophia.inria.fr

**Abstract**—We improve and validate TICIP [3], our TCP-friendly reliable transport protocol to collect information from a large number of Internet entities. A collector machine sends probes to a set of information sources that reply by sending back their reports. TICIP adapts the sending rate of probes in a way similar to TCP for the purpose of avoiding network congestion and implosion at the collector. Lost reports are requested again by TICIP until they are correctly received by the collector. In a first part of this work, we add to TICIP a mechanism to cluster information sources in order to probe sources behind the same bottleneck together. This ensures a smooth variation of network conditions during the collection session and hence, an efficient handling of congestion at network bottlenecks. We run simulations in ns-2 over realistic topologies to compare TICIP before and after clustering. We also implement the protocol in C++ and test it over the PlanetLab platform. All experiments prove the outperformance of TICIP over non adaptive solutions and the interest of the clustering mechanism in shortening the duration of the collection session and in decreasing the ratio of lost packets. In a second part, we adapt TICIP to collect large amounts of information from each data source. By the means of simulations, we compare the performance obtained by TICIP to that obtained when the information maintained by the different sources is collected by parallel TCP connections. Again, the simulations show that TICIP yields shorter collection sessions due to its inherent multiplexing capability. Finally, in a last part, we study the impact of delegating collection to some proxy sources that collect from other sources on behalf of the collector and that the collector probes later to get their collected data. We explain our method to choose the proxy collectors and we show by simulations that for a judicious choice of proxy collectors, one can decrease considerably the collection session duration.

## I. INTRODUCTION

TICIP [3] is a TCP-friendly reliable transport protocol designed to collect information from a large number of sources distributed throughout the Internet. It is a general purpose transport protocol that does not impose any constraint on the type of the collected data. This generality widens the spectrum of application of the protocol. TICIP can be used in all scenarios where entire data collection is needed. The data to be collected can be the availability of network entities, statistics on hosts, routers and network paths, quality of reception in a multicast session, numbering of population, weather monitoring, vote results, etc. TICIP is of particular interest in the area of network monitoring and topology inference (e.g., [4]) where the number of network entities

involved is large (hundreds of thousands of machines) and where it is preferable to end up the measurement in a relatively short time without incurring additional load on the network. TICIP would be the most suitable congestion and error control protocol to be used in this area to adapt the rate of probe packets as the pings and the traceroutes.

In TICIP, a collector machine sends probes to information sources, which reply by sending back report packets containing their information. Some difficulties come into play when designing a protocol like TICIP:

- There is a risk of network congestion due to bandwidth limitation and the large number of sources. Furthermore, all sources are not behind the same bottleneck which makes the congestion control more difficult.
- The collection traffic can be aggressive towards traffic generated by other applications. In particular, attention should be paid to prevent the collection traffic from penalising the concurrent TCP traffic.
- The loss of probes or reports lengthens the duration of the collection session, which urges for an efficient retransmission scheme.

TICIP does not only adapt the probing rate as a function of network conditions, but also tries to minimize the collection session duration by implementing an efficient retransmission strategy. Moreover, TICIP shares network resources fairly with concurrent traffic, namely TCP traffic, by adapting its probing rate in a similar way TCP.

TICIP was first introduced in [3]. It answers a need for a reliable transport solution to collect information in large IP networks. The present work enhances the protocol with three additional mechanisms to help it achieve better performances and adapt to more situations. This is in addition to a deep analysis of the protocol by the means of extensive simulations and real experiments over the PlanetLab platform [10].

The collector in the former version of TICIP [3] probes information sources in a random order. We start by showing that this strategy causes many problems when moving into large networks, which results in longer collection sessions, higher loss ratios and out of control traffic. The reason is

that only one control at the TICIP collector is used to limit the traffic at several bottlenecks simultaneously, which is clearly suboptimal given that congestion of one bottleneck can be hidden by the low utilization of another bottleneck and vice versa. To probe sources behind the same bottleneck together and separately from other bottlenecks, we add to TICIP a mechanism to gather information sources into clusters. This mechanism is based on the modelling of the Internet by a two-dimensions Euclidean space and its decomposition into clusters. We use to this end the Global Network Positioning system (GNP) [6] that provides Internet host coordinates. Our new mechanism makes it possible to traverse sources from the closest cluster to the collector in terms of RTT (Round Trip Time) to the farthest one. This very probably results in sources behind the same bottleneck probed together before the collector moves to neighbouring sources located behind another bottleneck. It is supposed that this behaviour improves the efficiency of the congestion control and ensures a smooth variation of its variables in TICIP.

To evaluate the performances of the protocol thus obtained, we run simulations with the ns-2 simulator [8] over realistic and complex network topologies. These simulations show that TICIP with the new mechanism of clustering has better performances than without clustering, and that it outperforms other non adaptive data collection solutions. To generalize this result, we implement the protocol in C++ in linux and run real experiments in the PlanetLab testbed. This practical experience tells us that TICIP can be easily deployed in the Internet and that the outperformance of the protocol observed in simulations is real.

Another challenge in developing a protocol like TICIP is how to extend it to the case of large information to be collected from each source. This information can be present at the moment the collection is initiated or can be generated later in sources. The old version of TICIP ensures collecting only one packet from each information source. In this paper, we explain how we have extended TICIP to support large size information collection and we evaluate this extension. The hard issue in this extension is how to handle acknowledgments and sequence numbers of packets and how to enable parallel collection so that to fully utilize the available bandwidth. We implement our extension to TICIP into ns-2 and we compare its performance to that of a collection application based upon parallel TCP connections. Simulation results show that TICIP has shorter collection session duration than parallel TCP connections and this is for any number of parallel TCP connections used. One can explain this result by the fact that TICIP multiplexes better the packets coming from all sources.

Afterwards, we move into studying the impact of delegating collection to a set of sources, called proxy collectors, on TICIP performance. A proxy collector plays the role of a TICIP collector from the viewpoint of sources on its charge, and plays the role of a regular information source (with large

volume of data) when seen by the main collector. Thus, there is a global TICIP session connecting the main collector to proxy collectors and local TICIP sessions connecting the proxy collectors to sources on their charges. The goal of this delegation is to minimize the duration of the collection session by replicating the collector probing functionality inside the network. The paper explains the method used to choose the proxy collectors as well as the changes made in TICIP to support this mechanism of delegation. With the help of ns-2, we vary the number of proxies in different scenarios and we note each time the duration of the collection session. As expected, results show that delegation improves performances. However, they also show the existence of an optimal number of delegated proxies otherwise we get a negligible gain compared to the no delegation case.

The paper is organized as follows. In the second section, we describe the main functionalities of TICIP [3]. We explain in Section III our approach to cluster information sources and we discuss therein simulation and experimental results. Section IV shows how we have extended TICIP to support the collection of large amount of information from each source together with the comparison by means of simulations to the case of parallel TCP connections. In the fifth section, we present the changes made in TICIP to ensure collection delegation and we discuss the choice of proxy collectors. Section VI summarizes the related work. The paper ends in Section VII with some conclusions and perspectives on our future research in this area.

## II. TCP-FRIENDLY INFORMATION COLLECTION PROTOCOL

TICIP [3] is a reliable transport information collection protocol implementing diverse functionalities. We focus here on those related to error recovery and network congestion control.

### A. Error recovery

The TICIP collector has a list of all information sources. Every source is distinguished by an identifier that can be for example its IP address. Sources whose reports are lost are probed again and a required to retransmit them until they are correctly received by the collector. To make the retransmission of reports in TICIP efficient, the collection session is made as a succession of rounds. In the first round, the collector sends request (probe) packets to all sources following their ranking in the list. In a second round, the collector sends requests to sources whose reports were not received in the previous round. The collector continues in rounds until it receives all reports. This behaviour in rounds is meant to wait for transitory network congestion to disappear from one round to another and to absorb the excessive delay that some reports may experience.

### B. Congestion control

To control the rate of requests and reports across the network, TICIP is based on a report-clocked window based

congestion control similar to the TCP one [2]. The collector maintains one variable  $cwnd$  indicating the congestion window size in number of requests/reports. New requests are transmitted only when the number of expected reports  $pipe$  is less than  $cwnd$ . TICIP adapts  $cwnd$  to the observed loss rate of reports. It proposes two algorithms to do so: Slow start and Congestion Avoidance.

1) *Slow start*: The collector starts a collection session by setting  $cwnd$  to  $RS$  (protocol parameter) and sending  $RS$  request packets. After some time, reports start to arrive. Some of these reports come on time, others are delayed. A timely report indicates that the network is not congested and that the collector can continue increasing its congestion window:  $cwnd = cwnd + 1$ . This yields a doubling of the probing rate for every window size of probes. The window continues growing in this way until the network becomes congested. At this point, the collector divides its congestion window by two and enters the congestion avoidance phase. The protocol comes back to slow start whenever a severe congestion appears (to be defined later).

2) *Congestion avoidance*: The congestion avoidance phase represents the steady state of TICIP. During this phase, the collector increases slowly  $cwnd$  in order to probe the network for more capacity. We aim a linear increase of the congestion window by  $RS$  probes every window size of probes. Thus, upon each timely report, the congestion window is increased by:  $cwnd = cwnd + \frac{RS}{cwnd}$ . When congestion is detected,  $cwnd$  is divided by two and a new congestion avoidance phase is started.

### C. Congestion detection mechanism

TICIP implements a congestion detection mechanism to compute report loss rates and to decide whether a report is on time, delayed or lost. This mechanism is based upon a timer  $TO$  scheduled at the beginning of the session and rescheduled again every time it expires.

1) *Round-trip time estimator*: TICIP sets the timer of the mechanism to an estimate of RTT (Round Trip Time), using the samples of RTT seen so far. The value of the timer is computed using estimates of the average RTT and of its variance. Let  $srtt$  and  $rttvar$  be the estimates of the average and the mean deviation of the RTT. Let  $rtt$  be the measured round-trip time when a report arrives. The collector updates the estimates and the timer ( $TO$ ) in the following way :

$$\begin{aligned} rttvar &= \frac{3}{4}.rttvar + \frac{1}{4}.|srtt - rtt| \\ srtt &= \frac{7}{8}.srtt + \frac{1}{8}.rtt \\ TO &= srtt + 4.rttvar \end{aligned}$$

This dynamics and the coefficients it involves are inspired from TCP.

2) *Detecting network congestion*: TICIP computes the report loss ratio during a time window equal to  $TO$ . When the timer is scheduled, the collector saves in the variable  $torecv$  the number of reports to be received before the expiration of the timer. Let  $recv$  be the number of timely reports received between the scheduling of the timer and its expiration. The

collector considers then that  $torecv - recv$  reports were lost in the network. Consequently, it estimates the loss ratio to  $1 - \frac{recv}{torecv}$ .

The network is considered congested if the loss ratio exceeds the Congestion Threshold ( $CT$ ) and severely congested if the loss ratio exceeds a higher threshold  $SCT > CT$  called the Severe Congestion Threshold.  $CT$  and  $SCT$  are two parameters of the protocol. TICIP set them as follows:

$$\begin{aligned} CT &= \min(0.1, \frac{RS}{cwnd}) \\ SCT &= \max(0.9, \frac{cwnd - RS}{cwnd}) \end{aligned}$$

Based on these values, network congestion for TICIP means that more than  $RS$  reports were lost in a window size of probes, while severe congestion means that less than  $RS$  timely reports were received.

3) *Delayed and timely reports*: A timely report is a report received before its deadline. The deadline of a report is given by the timer. A report not received before its deadline is assumed to be lost. If it arrives later than the deadline, it is considered to be delayed.

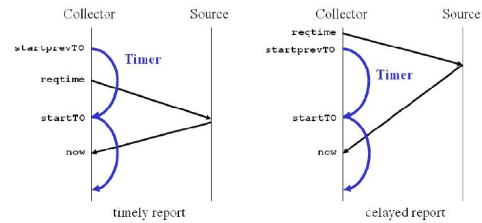


Fig. 1. The two types of reports

Figure 1 explains how the deadline of a report is set. Let  $startTO$  be the scheduling time of the timer. Let  $startprevTO$  be the previous scheduling time of the timer. When a report is received, the collector extracts from its header the timestamp  $reqtime$  indicating the time by which the corresponding probe has been sent. The report is received on time if and only if  $startprevTO < reqtime$ . The report is a delayed one in the opposite case.

### III. CLUSTERING OF INFORMATION SOURCES

The congestion control mechanism of TICIP described above relies on RTT estimation for the calculation of the report loss ratio and for the setting of deadlines to retransmit lost reports. In case sources are probed randomly and independently of their locations, this estimation is errored because of the high variability in the RTT process and its low auto-correlation. The loss ratio and report deadlines are then badly calculated which impairs the performance of the collection. In this section, we study the impact of clustering information sources on the performance of TICIP. The main idea is that the more sources are close to each other, the more likely they face the same network conditions on their paths to the collector. Close sources are very probably located behind the same bottleneck.

By clustering sources and probing clusters one after the other from the closest to the collector to the farthest one, one can ensure smooth variation of the RTT and in consequence, good estimation of the loss ratio and more precise deadlines. This allows handling bottlenecks separately from each other, which certainly results in better probing rate and shorter collection session duration. First, we detail the reasons that motivated the addition of the clustering mechanism. Then, we describe briefly the GNP Internet coordinate system[6] [5] that provides the knowledge of sources locations and we explain our approach to cluster information sources. Finally, we validate TICP with clustering through simulations and real experiments.

#### A. Need for clustering of information sources

Does a random ordering of sources yield a good estimation of RTT for the next pairs of request/report? The accuracy of this estimation is essential for the correct functioning of the protocol. An overestimation of RTT results in a delay in the detection of network congestion; the collector waits more than necessary for already lost reports. This delay means a waste of time and an aggravation of network congestion since the probing rate will not be reduced on time. On the other hand, an underestimation of RTT can cause errors in the computation of the report loss ratio due to the premature expiration of the timer. Thus, some reports can be declared lost while they are not. If it is the case, TICP will reduce unnecessarily the size of the congestion window (*cwnd*) which will lengthen unnecessarily the collection session.

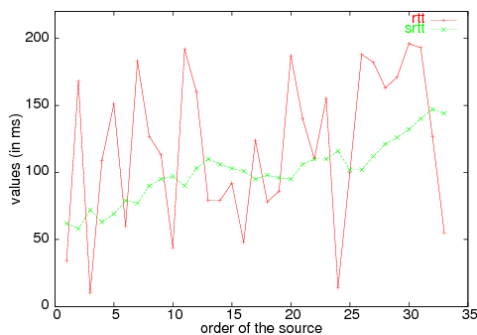


Fig. 2. Estimating RTT with random ordering of sources

To clarify this point, we plot in Figure 2 a sequence of *rtt* and *srtt* for sources picked randomly from the list of sources without any consideration of topology. The x-axis shows the probing order. Results are obtained from the ns-2 simulations to be described later. The figure clearly illustrates how the real value of the RTT oscillates; it can exceed the estimated average by large values and go much below it from time to time. This certainly implicates a regression in the performances of TICP. Our idea is to cluster information sources so as to probe them from the nearest to the collector to the farthest one. In this case the RTT in the network will vary smoothly and can be estimated more accurately. Figure 3 supports this claim. It

plots *rtt* and *srtt* when sources are ordered based on their latency to the collector. We can clearly observe how the RTT and the *srtt* vary smoothly with a low deviation from each other that can be easily compensated by the estimate of the mean deviation *rttvar*.

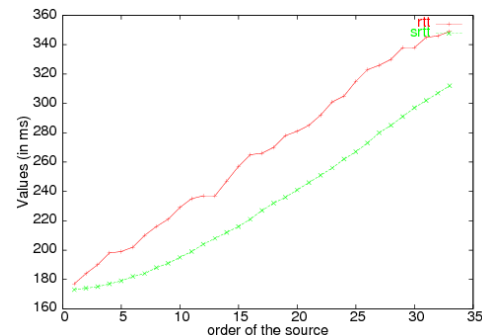


Fig. 3. Estimating RTT with latency-based ordering of sources

There is another gain from clustering. If the collector probes the sources in a random order, the request packets and the reports will circulate everywhere in the network. Hence, the collection traffic will participate in the congestion of several bottlenecks at the same time. But the collector cannot with one control adapt the rate of sending probes as a function of network conditions on all paths. In case of random ordering, the Internet is viewed by TICP as a sole bottleneck which is far from the reality. TICP should treat the bottlenecks one by one. The gathering of information sources into clusters and their probing from the nearest to the farthest ensures this decoupling of network bottlenecks and adapts correctly the probing rate to these bottlenecks one after the other instead of a joint adaptation.

#### B. Global Network Positioning System (GNP)

To support the clustering mechanism, we need the knowledge of hosts locations. Global Network Positioning (GNP) [6] [5] System is a promising and accurate solution to this problem. It models the Internet as an Euclidean space  $S$  of dimension  $n$ . This space has a well-defined coordinate base and a distance function. An internet host is represented in this space by a point such that the Round-Trip Time between two hosts can be estimated by computing the distance between their coordinates. To assign coordinates to hosts, GNP first computes the coordinates of a small set of well-known hosts called *landmarks* based on the measurement of delay among them. The coordinates of landmarks are then disseminated to any ordinary host willing to compute its own coordinates. An ordinary host derives its own coordinates based on landmarks coordinates and the measurement of delay between itself and the landmarks.

More technically, GNP is composed of two types of operations: landmarks operations and ordinary hosts operations. During the landmarks operations, the objective is to compute the coordinates  $C_{L_1}^S, C_{L_2}^S, \dots, C_{L_N}^S$  of the  $N$  landmarks  $L_1, L_2, \dots, L_N$  that minimize the sum of quadratic relative errors



domains. We assign latencies of 35ms for intra-transit domain links, 10 ms for stub-transit links and 5ms for intra-stub domain links. Figure 5 shows an example of a TS topology. The size of each topology realization is depicted by the parameter values in Table I. In each simulation we generate a TS topology and we choose randomly 500 sources of information and a collector among the nodes composing the topology. The parameters of TICP are set as in Table II. In particular, probe packets are of size 100 bytes and reports can fit in packets of 1500 bytes. With these values, congestion appears on the return path from sources to collector. As for the cluster size  $a$ , we set its value to 50 ms as our simulations indicate that this value leads to the best results over our TS topologies. Later in the paper, we change the value of  $a$  and we study how this impacts the collection sessions.

TABLE I  
TRANSIT-STUB MODEL PARAMETERS

Parameter	Signification	Scenario
T	Number of transit domains	5
Nt	Average Number of nodes / transit domain	7
K	Number of stub domains / transit node	8
Ns	Average number of nodes / stub domain	7

TABLE II  
TICP PARAMETERS

Parameter	Value
RS	10
probe size	100 B
report size	1500 B
$a$	50 ms

We compare between the both versions of TICP with and without clustering of information sources. The comparison criteria are network congestion and duration of the collection session. Figure 6 illustrates an example of the evolution of the congestion window size ( $cwnd$ ) as a function of time for TICP without clustering. First, we notice the saw tooth behaviour of TICP which adapts the window size to network conditions using the information on the loss ratio. But, we also notice that at time 17s, there was a reset of  $cwnd$  to  $RS$  following a severe network congestion (loss rate  $> SCT$ ). With random probing, TICP is unable to adapt the probing rate to the available bandwidth in several bottlenecks simultaneously. Figure 7 plots the same result but this time for TICP with clustering. It is clear that in this case TICP remains in the congestion avoidance phase and that the severe congestion does not appear. This illustrates that TICP with clustering adapts the window size to its right value without overwhelming the network. One can see TICP with clustering as treating bottlenecks one by one rather than at once. Note in the figures the decreasing trend in the window size, which is the result of the probing of

sources from the closest to the collector to the farthest from it.

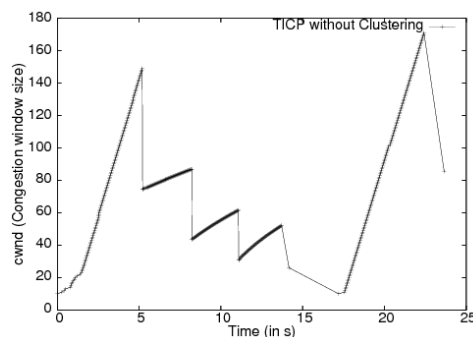


Fig. 6. Cwnd as a function of time for TICP without clustering

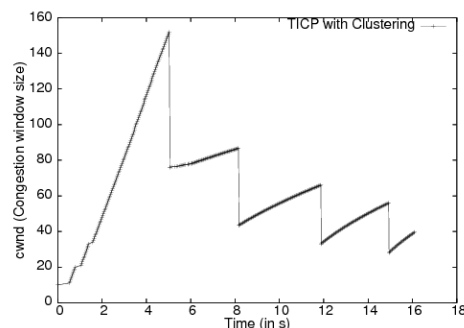


Fig. 7. Cwnd as a function of time for TICP with clustering

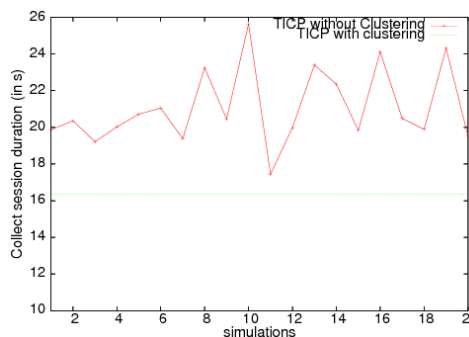


Fig. 8. Collection session duration for different ordering of sources

We continue the comparison between the two versions of TICP. This time we concentrate on the collection session duration. Figure 8 shows this duration for several simulations of TICP without clustering. In each simulation, the order of sources in the list of the collector is different, that is why we obtain each time a different duration for the collection session. For TICP with clustering, the result is the same since the topology does not change, and so the ranking of the sources is the same defined by their coordinates. TICP with clustering finds the good ranking of information sources and guarantees the shortest collection session duration. We

save on average 30% of time by moving from TICP without clustering to TICP with clustering.

To evaluate the optimality of TICP more generally, we implement in ns-2 an information collection protocol having a constant window size. For each window size, we run 10 simulations without clustering of sources and we record the minimum of the collection session duration over them in order to track the best combination. Figure 9 presents the evolution of this duration as a function of  $cwnd$ . The curve has a parabolic convex shape. For small congestion window sizes, the collection session is long because we have a low probing rate. For large window sizes, the network is congested which lengthens the collection session duration. The role of TICP is to find in a dichotomic way the right congestion window size that minimizes the collection session duration. We notice in the figure how TICP with clustering manages to reach the optimum unlike TICP without clustering which yields longer durations.

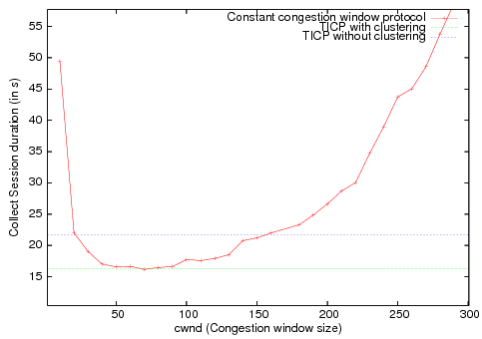


Fig. 9. Optimalty of the protocol

Then, we vary the cluster size and we study its impact on the collection session duration. Taking a very large  $a$  is equivalent to TICP without clustering since sources will be probed independently of their locations within the large cluster. Taking  $a$  very small results in clusters empty or with few number of sources. This is not efficient since there will be no clustering of sources behind common bottlenecks. So, there should be some average  $a$  that provides the best performance. Figure 10 validates this intuition where we can see that over the network topologies we consider in this work, a value of  $a$  around 50ms is optimal. Each point of the curve in the figure is the average over 5 simulation runs on different network topology realizations, all satisfying the characteristics in Table I. The number of sources is taken equal to 500.

Figure 11 studies how the number of sources impacts the choice of optimal  $a$ . We can clearly see that the optimal cluster size decreases when the number of sources increases. Compared to the value used above, the optimal  $a$  is equal to 85ms for 300 sources and to 45ms for 700 sources. Indeed, for small number of sources, one needs to increase  $a$  to group

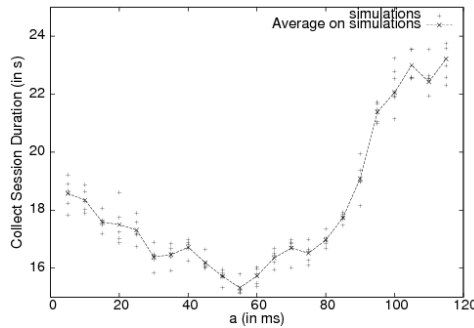


Fig. 10. Impact of  $a$  on collection session duration

more sources behind the same bottleneck together. At the opposite, for more sources, one needs to decrease  $a$  so that the collector can better probe them depending on their locations. But, if we continue increasing the number of sources, the optimal  $a$  will stabilize and become equal to some minimum value depending on the topology. One can safely use this value for applications collecting data from a very large number of sources. We suggest that in reality, one calculates this value by running multiple collections when TICP is used for the first time, then adapts it as a function of the measured session duration to account for any change in the underlying network topology.

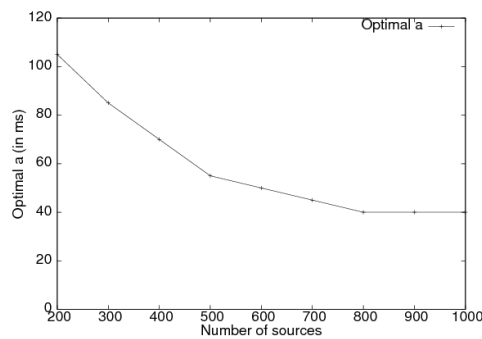


Fig. 11. Optimal  $a$  as a function of the number of sources

2) *Experimental results:* We implement TICP with clustering in C++ and run real experiments in the PlanetLab testbed [10]. We use one node from those available in our PlanetLab slice as collector and the other nodes as information sources. These experiments on PlanetLab aim to study the efficiency of deploying TICP in the real Internet and to validate the simulation results already obtained. We run successfully all scenarios described in our simulations. For lack of space, we include two samples from the results. Figure 12 plots the evolution of the collection session duration as a function of the number of sources for the optimal cluster size, and Figure 13 plots the evolution of the optimal cluster size as a function of the number of sources.

Both figures show that experimental results have the same shape compared to simulation ones but clearly different values

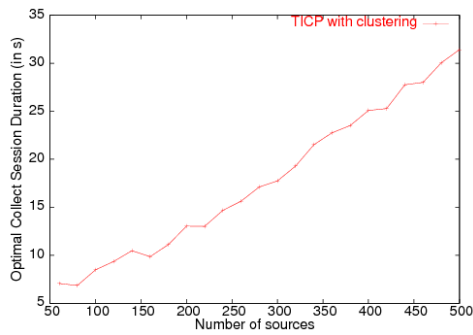


Fig. 12. Collection session duration as a function of number of sources in PlanetLab experiments

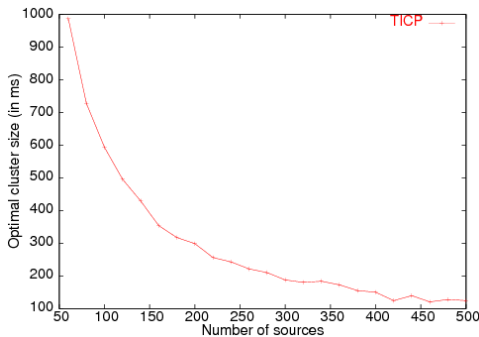


Fig. 13. Optimal cluster size as a function of number of sources in PlanetLab experiments

of the optimums. This is because the Internet topology is different from the topologies used in our simulations. The main observation we can make from Figure 12 is that the collection session duration grows almost linearly with the number of sources which means that TICP scales well in practice. As for Figure 13, it gives an idea on how to choose the cluster size when collecting information in the real Internet. It seems that clusters of 100ms side length lead to the best performances for a large number of Internet sources.

#### IV. COLLECTING LARGE AMOUNTS OF DATA

The first version of TICP [3] supposes that each information source has a small amount of data to send to the collector that can fit in one packet. This is a strong assumption that needs to be relaxed. Some sources might possess more than one packet when the session is initiated, or data that can be generated later during the collection session. In this section, we present our extension to the TICP protocol that allows collecting informational reports of several packets. We explain first the new functionalities added to TICP. We study then the scalability of the new version through simulations. Finally, we compare TICP to a collection application based on parallel TCP connections while varying the number of TCP connections.

This extension is far from being an easy task. First, it requires sequence numbers to detect the lost reports from each information source and to ask for retransmissions. Second,

it requires parallel collection from each information source when possible to better utilize the network resources. Lastly, it should adapt to the case when information is generated by the sources during the collection session. As to be described later, our extension handles all these issues in an efficient way.

3) *Behaviour of information sources:* An information source subdivides data into packets of equal sizes. Each packet has a unique sequence number *SEQ*. When the collector probes an information source for packet number *N*, the source sends back a data packet having *SEQ* equal to *N*. In general, the role of sources is simple as before and consists in answering the requests for packets sent by the collector.

An information source does not necessarily know the number of packets to send to the collector at the beginning of the session. We want to account for the scenario when data is continuously generated during the collection session. To indicate to the collector that the current packet is the last one, an information source sets a field *MORE* in this packet to 0. When *MORE* is equal to 1, the collector understands that the source has more packets to send and that it should keep probing it for more packets.

4) *Behaviour of the collector:* The collector probes sources in the order given by the clustering mechanism. The sending rate of probes respects the congestion window size as before.

In the first round, the collector probes all sources asking them to send their first packet. The requested packet is the packet having *SEQ* equal to 1. To make this possible, we add to the request packet a field *RSEQ* indicating the sequence number of the requested packet. In the second round, the collector will discover that some packets requested during the first round have arrived to the collector while others have been delayed or lost. The probing scheme will be to ask a source which has correctly sent her first packet to send its second one and to ask a source whose first packet has been declared to be lost to retransmit it. As for sources whose first packet has not yet arrived (their deadlines have not expired), an efficient solution will be to ask for their second packet, of course if the congestion window allows and if the field *MORE* has not been set to 0 for these sources. By behaving in this way, TICP is able to anticipate the arrival of packets and fill the congestion window which is necessary for an efficient utilization of network resources. This behaviour is generalised to the following rounds.

To implement this behaviour, we introduce the following structure at the collector that contains all required information about the sources during the collection:

```

SOURCES_LIST = {SOURCE}
SOURCE = [ORDER, ID_SRC, MORE, REQUESTS,
RECEIVED_PACKETS]
REQUESTS = [LAST_SENT_REQUEST, REQUESTS_LIST]
RECEIVED_PACKETS = [LAST_RECEIVED_PACKET,
PACKET_NUMBERS_LIST]
REQUESTS_LIST = {REQUEST}
PACKET_NUMBERS_LIST = {SEQ}
REQUEST = [RSEQ, DEADLINE]

```

*SOURCES\_LIST* is the list of information sources. It is composed of objects of type *SOURCE*. This list is ordered following the clustering mechanism previously described. The order of a source is stored in the *ORDER* field. The attribute *MORE* equals 1 if the source still has packets to send to the collector. It is equal to 0 if the source has successfully sent all its packets and the collector has received the last packet carrying the field *MORE* equal to 0.

The attribute *REQUESTS* is an object giving an idea on the progress of the probing processes of this source. It is composed of two attributes *LAST\_SENT\_REQUEST* and *REQUESTS\_LIST*. *LAST\_SENT\_REQUEST* indicates the highest sequence number of already sent requests. *REQUESTS\_LIST* is a list of objects of type *REQUEST*. A *REQUEST* object is composed of the sequence number of the request *RSEQ* and of its transmission time *DEADLINE*. The latter attribute allows the collector to decide whether the deadline of the corresponding data packet has been reached or not.

The attribute *RECEIVED\_PACKETS* is an object allowing the collector to follow the progress of the reception of data packets from the source. In fact, packets sent by one source do not always arrive in order. It is then important to store in a variable *LAST\_RECEIVED\_PACKET* the sequence number of the last in-order packet received from this data source and in an object of type *PACKETS\_NUMBERS\_LIST* the sequence numbers of packets not received in order.

When it is the turn of an information source, the collector checks the *MORE* field of the object *SOURCE* corresponding to this source. If *MORE* equals 0, then the collector jumps over this source and treats the next source in the list. If all sources have the *MORE* field equal to 0, the collector announces the end of the collection session. In the case when the *MORE* field of a source is still equal to 1, the collector first seeks in the list of sent requests related to this source a request packet whose deadline has been reached. We recall that the deadline of a request/data packet is reached when *DEADLINE* is less than *startprevTO*, the time at which the previous timer has been scheduled (see Section II). If such request is found, the collector considers that the corresponding report is lost, cancels this request, retransmits it and jumps to the next source in the list. If *REQUESTS\_LIST* is empty or the collector has not found any request whose deadline has been reached, it sends a request packet having *RSEQ* equal to *LAST\_SENT\_REQUEST* + 1. This means that it anticipates requesting the next packet and gives more time to delayed packets to arrive at the collector.

At the beginning of the session, *LAST\_SENT\_REQUEST* is set to 0. Any newly sent request must be added to *REQUESTS* and *LAST\_SENT\_REQUEST* must be incremented. In the case of a retransmitted request, a request packet having the same *RSEQ* is sent and *DEADLINE* is set equal to the current time. The attribute *LAST\_SENT\_REQUEST* does not change in this case.

When the collector receives a data packet, it updates *cwnd* and *pipe* as described earlier (see Section II), then it deletes the corresponding request from *RE-*

*QUESTS\_LIST*. Finally, it updates *PACKETS\_NUMBERS\_LIST* and *LAST\_RECEIVED\_PACKET*.

#### A. Scalability of the new version of TICIP

We want to test the scalability and the efficiency of this new version of TICIP when the number of packets per source and the number of sources grow. For this, we run simulations in ns-2 on the same network topology used to validate the clustering approach. The sizes of a request packet and a data packet are respectively taken equal to 100 bytes and 1000 bytes.

The first set of simulations fixes the number of sources and studies the impact of changing the number of packets per source on the performance of TICIP. Figure 14 plots the evolution of the collection session duration as a function of the number of packets per source for three values of the number of sources  $N1=50$ ,  $N2=100$ ,  $N3=150$ . We observe that for a

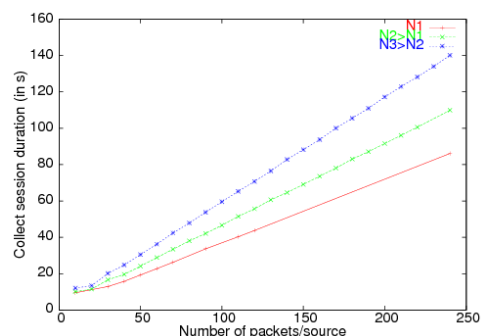


Fig. 14. Impact of the number of packets per source

given number of sources, the collection session duration varies linearly with the number of sources. The slope of the line seems to only depend on the number of sources. When the number of sources increases, this slope increases. This linear evolution proves the scalability of TICIP when the number of packets per source increases. One can interpret this behaviour by the fact that the network is seen by TICIP as a set of bottlenecks. The collection session duration is the sum of the collection times over the different bottlenecks. We know that the collection time of lets say  $S$  bytes of data over a bottleneck having  $B$  bps as available bandwidth, is almost linear with  $S$  and equal to  $\frac{S}{B}$ . When adding packets in sources, it is as if we are increasing linearly the number of packets to collect over each bottleneck, which results in this linear behaviour.

The second set of simulations fixes the number of packets per source and varies the number of sources. Figure 15 presents the evolution of the collection session duration as a function of the number of sources for three values of the number of packets per source (150, 200 and 250). The figure shows that the collection session duration varies almost linearly with the number of sources. Again, the interpretation is that the network is formed of a limited set of bottlenecks, and by adding sources randomly in the network, we are increasing proportionally the volume of data to collect over each bottleneck.

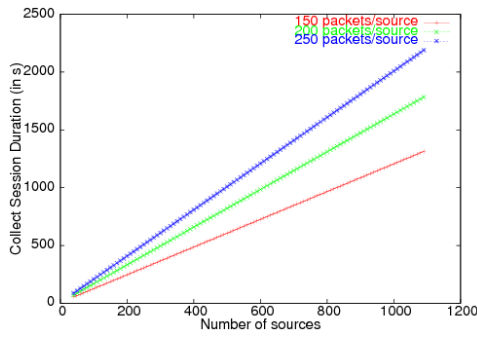


Fig. 15. Impact of the number of sources

### B. Comparison with a collection application using parallel TCP connections

The intuitive question that one would ask at this level is how TICIP compares to a collection application using regular TCP to collect data from sources (one TCP connection per source). For the case of small amount of data per source (say one packet), the answer to this question is the outperformance of TICIP for the simple reason that it avoids the three-way handshaking of TCP and that it allows the parallel collection from a large number of sources based on network conditions. For the case of large amounts of data, this outperformance is not evident and needs to be proven.

We compare here TICIP to a collection application using parallel TCP connections to gather packets from sources. One TCP connection is opened between the collector and each source to retrieve its information. The collection application using TCP limits the number of parallel connections to let's say  $m$  connections. When one connection ends, another connection is established with another source and this continues until all data is retrieved. The main challenge here is to well choose the number of parallel TCP connections that minimises the collection session duration. Several works have studied the problem of how many parallel TCP connections one needs to open to get the best performances and all agree on that only few connections need to be established in parallel. [1] shows through emulations that for a large number of connections, the network becomes congested and the throughput is low. On the other hand, for  $m$  very small, one does not use completely the available bandwidth and so the throughput of reception is not the best. [1] observes that the throughput reaches its maximum for a number of connections between 6 and 8.

Having this in mind, we design the collection application based on parallel TCP as follows. The application begins by opening  $m$  connections from the collector to the first  $m$  sources in the list. When a connection finishes retrieving packets from the source to which it connects, another connection is opened with the next available source in the list. The application continues in this way until reaching the last source in the list.

To compare with TICIP, we implement this collection application in ns-2 and we generate topologies similar to those used earlier in this paper. We choose 500 sources and a collector randomly among the nodes of the topology.

Each information source is supposed to have 200 packets of 1000 bytes each to send to the collector. First, we run TICIP under this configuration. Then, we run the collection application for different values of the number of parallel TCP connections  $m$ . Figure 16 plots the evolution of the collection session duration as a function of the number of parallel TCP connections. It also shows the collection session duration for TICIP.

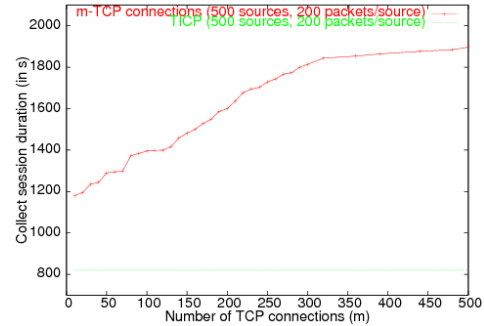


Fig. 16. Parallel TCP connections Vs. TICIP

As expected, the figure shows that a small number of parallel TCP connections gives the optimal collection session duration. But surprisingly, the figure also shows that the collection session duration for TICIP is shorter than the collection session duration for any number of parallel TCP connections. This proves the efficiency and the importance of TICIP for the collection of even large amounts of data from a large number of sources. We interpret this result by the fact that in case of TCP, at least one slow start phase is needed by each connection, which wastes network bandwidth. In case of TICIP, these slow starts are avoided since the collection is done in parallel from all sources with only one congestion window. The other explanation is that TICIP collects data in parallel from all sources, which allows an efficient utilization of network resources. In case of parallel TCP, we are obliged to open connections with a limited number of sources and collect from them entirely before moving to other sources. This is suboptimal since only few bottlenecks are utilized while others are not.

## V. PARALLELIZING THE COLLECTION USING PROXY COLLECTORS

Even though TICIP has been shown to be efficient, its performances are still limited by the fact that there is only one collector that probes sources distributed worldwide. In this section, we study the impact of delegating information collection to a set of proxy collectors. The goal of this delegation is to decrease the collection session duration. Proxy collectors are specific well-dimensioned machines responsible of collecting packets from sources in their neighborhood. In addition, we assume the more general scenario where proxy collectors are sources of information themselves. The main collector in its turn is responsible of the global collection by gathering packets

from proxy collectors. First, we present the method we use to choose the proxy collectors among information sources. Then, we explain how we extended TICP to support delegation. Finally, we discuss the simulation results.

#### A. Choice of proxy collectors

The idea is to choose  $p$  proxy collectors among the sources in a way to minimize the sum of the latencies from each source to its closet proxy collector. This way we minimize the length of network paths crossed by probes and reports. The choice of proxy collectors is a particular instance of the well-known  $p$ -median problem. The  $p$ -median problem is defined as follows. Given a set  $F$  of  $l$  potential facilities, a set  $U$  of  $n$  users, a distance function  $d: U \times F \rightarrow R$ , and a constant  $p \leq l$ , determine which  $p$  facilities to open so as to minimize the sum of the distances from each user to its closest open facility. This is a well-known NP-hard problem. In our case, the set of information sources represents the facilities as well as the users. The distance function is the Euclidean distance computed using sources' coordinates given by GNP.

To solve this problem, we use the hybrid heuristic proposed in [11]. It is a multi-start iterative method where each iteration consists of a randomized greedy construction of a solution, which is then submitted to local search. A pool of best solutions found in iterations is kept. In each iteration, the solution found with local search is combined with one of elite solutions. After all iterations, there is a post-optimization phase in which elite solutions are combined with each other. This method gives good quality solutions in short running times.

#### B. TICP with collection delegation

Three types of actors take part of a TICP session with collection delegation: the main collector, the  $p$  proxy collectors and the  $n-p$  ordinary information sources.  $n$  is the total number of sources. A collection session starts with choosing  $p$  information sources as proxy collectors using the above heuristic. Then, the main collector runs a TICP session between itself and the proxy collectors. In its first probe packets to a proxy collector, the main collector sends the identifiers of ordinary sources that are under the responsibility of this proxy collector. Once a proxy collector receives the list of sources for which it is responsible, it updates its data structure to begin immediately a local TICP collection session. During this local session<sup>1</sup>, it plays the role of a regular TICP collector towards ordinary sources in its charge. This behaviour is described with details in section IV.

Now from the point of view of the main collector, a proxy collector behaves like an information source. It considers the packets issued by ordinary sources under its responsibility as its own packets. First, it starts by sending its own packets to the collector. Then, it sends the packets of ordinary sources in the order of their arrivals. It may happen that the main collector asks the proxy collector to send a new packet

<sup>1</sup>It can be seen as level-2 session compared to the collectors one which can be seen as level-1.

and that the proxy collector is waiting for new packets to arrive from ordinary sources. In this case, the proxy collector replies with a *persistence* packet instead of the requested packet to tell the main collector that it has nothing to send right now but that more packets will come later. We add this persistence packet in order for the main collector not to consider the absence of a response as an indication of network congestion. Such a persistence packet has the same sequence number as the requested packet but with a *MORE* field set to 2.

The role of a proxy collector ends when it receives from each ordinary source in its charge an information packet having a *MORE* field equal to 0 and when it succeeds to send all packets it has received to the main collector. To indicate the end for the local collection carried out by the proxy collector, the last information packet sent to the main collector must contain a *MORE* field equal to 0. The global session ends when a *MORE* field equal to 0 is received from all proxy collectors.

We run simulations in ns-2 over network topologies described in Section III in order to evaluate the gain obtained when delegating the collection. We vary the number of proxy collectors and we compute for each number, the average duration of the collection session over several simulations. Figure 17 plots the results on the y axis with the x-axis showing the number of proxy collectors.

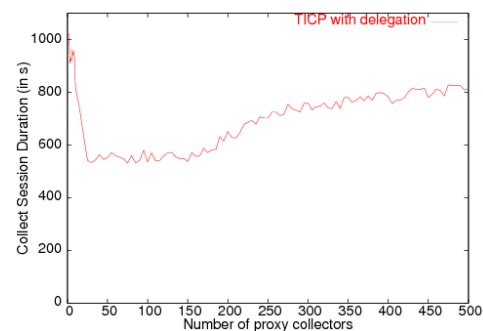


Fig. 17. TICP with delegation

We can clearly observe in the figure an improvement in the performances of TICP when we delegate to the first proxies, then a deterioration in the performances as long as we add more proxies. There is optimal number of delegated proxies that leads to the shortest session (a gain by two in our settings). Indeed, for a small number of proxy collectors, there is a waste of time and resources since the proxy collectors are in charge of many sources and so are overwhelmed and not able to answer in real time the probes of the main collector. If we continue increasing the number of proxies, TICP will be able to collect with a high speed from the proxy collectors who themselves are collecting at high speed from the ordinary sources, which explain the best performances. Further increase in the number of proxy collectors lengthens the collection session since the main collector becomes in charge of many proxy collectors which prohibits it from collecting from them

at high speed. One can understand this better by looking at the extreme and intuitive case of delegating to everyone which is equivalent to the case of not delegating at all.

Given the presence of an optimal value for the number of delegated proxies, we want to check how this optimal value varies with the number of ordinary sources. For this, we vary the number of sources and record the number of proxy collectors giving the minimal collection session duration. The results are average over several simulations and are plotted in Figure 18. We can see that when the number of sources is less than some threshold (140 in our settings), it is better to delegate to all sources (or equivalently not to delegate at all). In this case, the population is small and one TICIP session is able to do the job. For a number of sources larger than the threshold, we notice that the optimal number of proxy collectors to consider is almost constant equal to the threshold (140 in our settings). Our interpretation for this behavior is that there is maximum capacity for the network that can be achieved by some delegation and any further increase of sources beyond this delegation will be equivalent to adding more data per sources and so will not impact the number of delegated proxies. This maximal of delegated proxies to be used is topology dependent. Our future research will focus on its calculation for different scenarios.

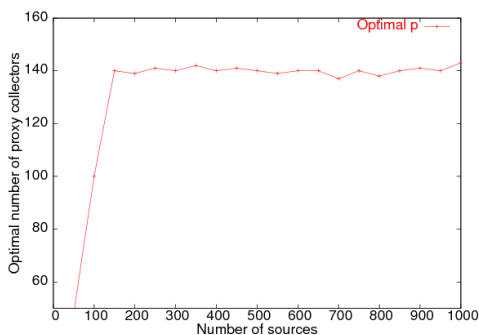


Fig. 18. Optimal number of proxy collectors

This figure shows that for a number of proxy collectors less than 140, the delegation of collection is not necessary since the number of optimal proxy collectors is equal to the number of sources. For a number of sources greater than 140, we notice that the values of optimal number of proxy collectors are almost equal to 140. In fact, the main collector can not gather information faster than the available bandwidth in the network. One must not forget that we are increasing the number of sources but not changing the topology. Hence, it is easy to choose the number of proxy collectors that minimizes collection session duration.

## VI. RELATED WORK

The collection of information has been studied in the literature in different contexts. We discuss in this section the two most relevant to our work: reliable multicast and sensor networks.

In reliable multicast, sources that did not receive a packet send a NACK asking the collector for a retransmission of the packet (we keep the terms collector and sources even though the collector in this context is transmitting data and the sources are listening). Many NACKs may cause congestion in the network or at the collector. The problem is called NACK implosion. But, the NACK information can be safely filtered; there is no need that a host sends a NACK if another source has already sent a NACK for the same packet, since the collector will retransmit anyway the lost packet to all members of the multicast session. The aggregation of NACKs can be done either along a tree that connects all sources or using multicast itself. In [13], it is proposed that leaf sources send their NACKs to a parent source (called designated receiver), which aggregates this information and sends it to its parent until it reaches the collector. In [14], a source waits for a random time before sending a NACK, and listens at the same time if another source has sent a NACK for the same packet. If so, the former source cancels its request, otherwise it sends it when the timer expires. Another approach for NACK aggregation is to use the principle of active networks to program nodes of a multicast tree as advocated by [15].

The reliable collection of information has also its application in the context of sensor networks. Sensors are sources of information that wake up generally when an event happens and send information about this event to some collecting point called the sink. Some protocols exist in the literature for a reliable collection, e.g., [16], [17]. These protocols agree on that end-to-end transport solutions lead to poor performance given the noisy nature of wireless links connecting the sensors, and the absence of permanent routes caused by the intermittent wake up of sensors and their limited lifetime. Per-hop transport protocols have been advocated for sensor networks. The information is proposed to be reliably sent from one sensor to another until it reaches the sink, with retransmissions done on a hop-per-hop basis. Clearly, such solutions are not optimal in the wired Internet where permanent routes exist and are provided by the IP protocol. Moreover, the round-trip time in the Internet is usually in the order of hundreds of milliseconds and links are of good quality. All this make an end-to-end solution the most appropriate for the Internet, which is what TICIP provides. Also note that losses in the Internet are mostly caused by congestion and that other traffic exists, so transport protocols have to implement end-to-end congestion avoidance and error control mechanisms to reduce the number of losses and to provide fairness. This is not the case in sensor networks where information collection is the major source of traffic.

## VII. CONCLUSION AND PERSPECTIVES

TICIP is a transport protocol to collect information from a large number of network entities. It aims to control the congestion of the network and to minimize the collection session duration. This work brings three major modifications to TICIP. First, to ensure a smooth variation of the congestion control parameters, we have added to TICIP a mechanism

to cluster information sources. Simulation and experimental results showed that this mechanism ameliorates the performances by reducing the loss rate of reports and leading to shorter collection session durations. Second, we extended the protocol to support collecting large amounts of data. With this modification, TICIP collector is able to collect several packets from each data source. We proved by simulations that our approach is scalable and that TICIP performs better than a collecting application using parallel TCP connections. Lastly, we enhanced the collection further by delegating it to some intermediate nodes called proxy collectors. We modified TICIP to support this delegation and to choose optimally the proxy machines. The simulations showed that for a well-chosen number of proxy collectors, TICIP is able to collect faster data from sources. The optimal number of proxies to be used seems to be topology dependent and not function of the number of sources of information.

Our future research will be about the experimental evaluation of TICIP in case of large amounts of data per source and delegated proxies. We are also working towards the integration of TICIP in measurement infrastructures for the purpose of controlling the rate of probing packets and for the collection of passive measurements carried out at different links of the network.

#### REFERENCES

- [1] Mark Allman, Hans Kruse, Shawn Ostermann. "An Application-Level Solution to TCP's Satellite Inefficiencies", Proceedings of the First International Workshop on Satellite-based Information Services (WOSBIS), Rye, New York, November 13, 1996.
- [2] M.Allman, V.Paxson, W.Stevens,"TCP Congestion Control", RFC 2581, April.1999.
- [3] Chadi Barakat, Mohamed Malli, Noamichi Nonaka,"TICIP:Transport Information Collection Protocol", in Annals of Telecommunications, vol.61, no. 1-2, pp. 167-192, January-February, 2006.
- [4] B. Donnet, P. Raoult, T. Friedman, M. Crovella, "Deployment of an Algorithm for Large-Scale Topology Discovery", in IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Internet Sampling, Dec. 2006. vol. 24, n. 12.
- [5] T. S. Eugene Ng and Hui Zhang, "Towards Global Network Positioning", Extended Abstract, ACM SIGCOMM Internet Measurement Workshop 2001, San Francisco, CA, November 2001.
- [6] T.S Eugene Ng and Hui Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches", INFOCOM'02,New York,NY,June 2002.
- [7] Ken Calvert, Matt Doar and Ellen W. Zegura, "Modeling Internet Topology, IEEE Communications Magazine, June 1997.
- [8] The Network Simulator ns-2, <http://www.isi.edu/nsnam/ns/>
- [9] V.Paxson, M.Allman, "Computing TCP's Retransmission Timer", Internet Draft, April 2000.
- [10] The PlanetLab platform, <http://www.planet-lab.org/>
- [11] M. G. C. Resende and R. F. Werneck, "A hybrid heuristic for the p-median problem, Technical Report TD-5NWRCR", AT&T Labs Research, 2003. Published in Journal of Heuristics, vol. 10, pp. 59-88, 2004
- [12] Ellen W.Zegura, Ken Calvert and S. Bhattacharjee,"How to Model an Internetwork". Proceedings of IEEE Infocom '96, San Francisco, CA.
- [13] Lin (J.C.), Paul (S.), RMTP: A Reliable Multicast Transport Protocol.IEEE INFOCOM, March 1996.
- [14] Floyd (S.), Jacobson (V.), McCanne (S.), Liu (C.), Zhang (L.), A reliable multicast framework for light-weight sessions and application level framing. ACM SIGCOMM, August 1995.
- [15] Calvert (K. L.), Griffioen (J.), Mullins (B.), Sehgal (A.), Wen (S.), Concast: Design and Implementation of an Active Network Service. IEEE Journal on Selected Area in Communications (JSAC), vol. 19, no. 3, March 2001.
- [16] Stann (F.), Heidemann (J.), RMST: Reliable Data Transport in Sensor Networks. IEEE International Workshop on Sensor Net Protocols and Applications (SNPA), May 2003.
- [17] Wan (C.), Campbell (A.), Krishnamurthy (L.), PSFQ: A Reliable Transport Mechanism for Wireless Sensor Networks. ACM International Workshop on Wireless Sensor Networks and Applications, September 2002.