Experiences on enhancing data collection in large networks

Mohamed Karim Sbai, Chadi Barakat

Experiences on enhancing data collection in large networks

Mohamed Karim Sbai, Chadi Barakat

Project-Team Planète, INRIA Sophia-Antipolis méditerranée, France
Email: Mohamed_Karim.Sbai@sophia.inria.fr, Chadi.Barakat@sophia.inria.fr

Abstract

We improve and validate TICP [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. TICP adapts the sending rate of probes in a way similar to TCP for the purpose of avoiding network congestion and keeps requesting reports until they are well received. In a first part of this work, we add to TICP a mechanism to cluster information sources in order to smooth the variation of network conditions during the collection session and to ensure an efficient handling of congestion at network bottlenecks. Simulations in ns-2 and PlanetLab experiments prove the outperformance of TICP 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 TICP to collect several packets of information from each data source. By the means of simulations, we compare the performance obtained by TICP to that obtained when using parallel short-lived TCP connections. Our main observation is that TICP yields shorter collection sessions due to its inherent multiplexing capability and that it avoids parallel Slow Start phases. Finally, in a last part, we study the impact of delegating collection to some proxy collectors. We explain our method for delegation and we show by simulations that for a judicious choice of proxy collectors, one can decrease considerably the collection session duration.

Key words: Transport protocol, information collection, clustering, several packets, short-lived TCP connections, delegation

⋆ This paper summarizes our contributions on the TICP protocol. It relies on our two previous publications [3] and [4] and goes beyond by presenting experimental results over PlanetLab and two new and important components, that are the extension to the multiple packets per source case and the delegation of the collector function to intermediate proxies.

1 Introduction

TICP [3] stands for TCP-friendly Information Collection Protocol. It is a TCP-friendly reliable transport protocol designed to collect information from a large number of sources distributed throughout the Internet. TICP is a general purpose end-to-end transport protocol that does not impose any constraint on the type of the collected data and that does not require network collaboration. This generality widens the spectrum of application of the protocol. TICP 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. It is of particular interest in the area of network monitoring and topology inference (e.g., [5]) where the number of network entities involved is large (hundreds of thousands of machines and routers) and where it is preferable to end up the measurement in a relatively short time without incurring additional load on the network. TICP is an appropriate congestion and error control protocol to be used in this area for the purpose of adapting the rate of probe packets as the pings and the traceroutes.

In TICP, 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 data collection protocol like TICP:

- 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.

TICP 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, TICP shares network resources fairly with concurrent traffic, namely TCP traffic, by adapting its probing rate in a similar way to TCP.

TICP was first introduced in [3]. It answers a need for a reliable transport solution to collect information in large IP networks. Current solutions used in network monitoring and management as in Skitter [17] or in the SNMP protocol [15] implement simple periodic probing at a low rate in order not the overload the network. For example in SNMP and to collect average traffic

statistics over 5 minute intervals, routers are probed periodically by no more than once every 5 minutes. For short intervals and large number of routers the overhead on the network can become significant. Other solutions like Concast [14] rely on routers to aggregate the collected information and so do not work in the current Internet.

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 [9]. The first mechanism has been introduced in [4]. The second and third mechanisms are novel. The implementation of the code in C++ to run over UDP sockets in the PlanetLab platform is also one of our new contributions in this area.

The collector in the former version of TICP [3] probes information sources in a random order. We start by showing that this strategy causes many problems when moving to large networks, which results in long collection sessions, high loss ratios and out of control traffic. The reason is that only one control at the TICP collector is used to limit the traffic at several bottlenecks simultaneously, which is clearly suboptimal given that the congestion of one bottleneck router can be hidden by the low utilization of another bottleneck router and vice versa. To probe sources behind the same bottleneck together and separately from other bottlenecks, we add to TICP a mechanism to gather information sources into clusters. This mechanism is based on the modeling of the Internet by a two-dimensional 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 probe sources from the closest cluster to the collector in terms of RTT (Round Trip Time) to the farthest one. This very likely results in sources behind the same bottleneck probed together before the collector moves to neighboring 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 TICP.

To evaluate the performances of the protocol thus obtained, we run simulations with the ns-2 simulator [7] over realistic and complex network topologies. These simulations show that TICP 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 on the PlanetLab testbed. This practical experience tells us that TICP 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 TICP is how to extend it to

the case of several packets of information to be collected from each source. This information can be present at the moment the collection is initiated or can be generated later by sources. The old version of TICP [3] ensures collecting only one packet from each information source. In this paper, we explain how we have extended TICP to support collecting several packets per source 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 TICP into ns-2 and we compare its performance to that of a collection application based upon parallel TCP connections. When the number of packets to be collected from each data source is not important, TCP connections will be short-lived. Simulation results show that TICP has shorter collection session than parallel short-lived TCP connections and this is for any number of parallel TCP connections used. One can explain this result by the fact that TICP multiplexes better the packets coming from all sources. Furthermore, three-way handshake is a problem in short- lived TCP connections and having several Slow Start phases in parallel is sub-optimal. TICP has only one slow start phase for all sources.

We believe that deploying TICP to collect very large amounts of data will not bring much compared to the use of parallel TCP connections. TICP was designed to collect one to several packets from each source. But one can note that even in the case of long-lived TCP connections, TICP does not require adjusting any number of connections since it has only one connection to all sources. Our simulation results will confirm this observation.

Afterwards, we move into studying the impact of delegating collection to a set of sources, called proxy collectors, on TICP performance. A proxy collector plays the role of a TICP collector from the viewpoint of sources on its charge, and plays the role of a regular information source (with several packets of data) when seen by the main collector. Thus, there is a global TICP session connecting the main collector to proxy collectors and local TICP 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. This paper explains the method used to choose the proxy collectors as well as the changes made to TICP to support this mechanism of delegation. With the help of ns-2, we vary the number of proxies in different scenarios and we measure the duration of the collection session. As expected, results show that delegation improves performances. However, we observe 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 Section 2, we describe the main functionalities of TICP [3]. We explain in Section 3 our approach to cluster information sources and we discuss therein simulation and experimental results. Section

4 shows how we extend TICP to support the collection of several packets of information from each source together with the comparison by simulations to the case of parallel TCP connections. In the fifth section, we present the changes made to TICP to ensure collection delegation and we discuss the choice of proxy collectors. Section 6 overviews the related work and Section 7 ends the paper with some conclusions and perspectives on our future research in this area.

2 TCP-friendly Information Collection Protocol

TICP [3] is a reliable transport information collection protocol implementing diverse functionalities. We overview in this section those related to error recovery and network congestion control.

2.1 Error recovery

The TICP collector has a list of all information sources. The way to obtain this list is out of the scope of TICP. Every source is distinguished by an identifier that can be for example its IP address. Sources whose reports are lost are probed again and are requested to retransmit their reports until they are correctly received by the collector. To make the retransmission of reports in TICP 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 behavior 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.

2.2 Congestion control

To control the rate of requests and reports across the network, TICP implements 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 or reports. *cwnd* is then the maximum number of requests that the collector can send without receiving any report packet. New requests are only transmitted when the number of expected reports *pipe* is less than *cwnd*. TICP adapts *cwnd* to the observed loss rate of

reports. It proposes two algorithms to achieve this objective: Slow start and Congestion Avoidance.

2.2.1 Slow start

The collector starts a collection session by setting *cwnd* to *RS* (Request Size, a protocol parameter) and sending *RS* request packets. [1] 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).

2.2.2 Congestion avoidance

This represents the steady state phase of TICP. During this phase, the collector increases slowly *cwnd* 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.

2.3 Congestion detection mechanism

TICP 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 every time it expires.

2.3.1 Round-trip time estimator

TICP sets the timer of the mechanism to a conservative estimate of RTT (Round Trip Time), using the samples of RTT seen so far. The value of the

[1] *RS* (Request Size) is the initial number of requests sent without receiving any report packet. It is also the step by which the window size increases. Compared to TCP, this can be seen as the packet size in bytes.

timer is computed using estimates of the average RTT and of its mean deviation. 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 :

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

This dynamics and the coefficients it involves are inspired from TCP [8]. The weights for the estimation of *srtt* and *rttvar* are known in TCP to be easy to calculate and to provide good balance between convergence and precision. Setting TO to *srtt* plus four times the *rttvar* has shown its efficiency in setting an upper bound on the Round-Trip Time in TCP. Since our estimator has almost the same target as in TCP, we adopt the same strategy.

2.3.2 Detecting network congestion

TICP computes the report loss ratio over a time window in the past 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 our protocol. They can be set to some default values, for example, to 10% for CT and to 90% for SCT. We set them as follows:

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

The minimum and maximum functions in the expressions of CT and SCT are necessary to ensure that these thresholds do not take unrealistic values when the congestion window is of small size (close to RS). One can use other default values than 0.1 and 0.9. Set as above, CT is equal to $\frac{RS}{cwnd}$ for large congestion windows, which means that congestion is concluded when more than RS reports are not received (resp. severe congestion is concluded when less than RS reports are received in a window). We recall that a report is not received if it is lost or delayed, or if the corresponding request itself is lost or delayed. This way we are compliant with TCP, which considers that the network is congested if at least one packet is lost, and severely congested

when all packets are lost or delayed (i.e., they arrive after the expiration of the timer). Note that we are designing TICP so that a packet in TCP is comparable to *RS* requests/probes in our context. A TICP session behaves like this as *RS* TCP connections running in the same environnement, a parameter that the administrator can tune to whatever desired value.

2.3.3 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 its deadline, it is considered to be delayed.

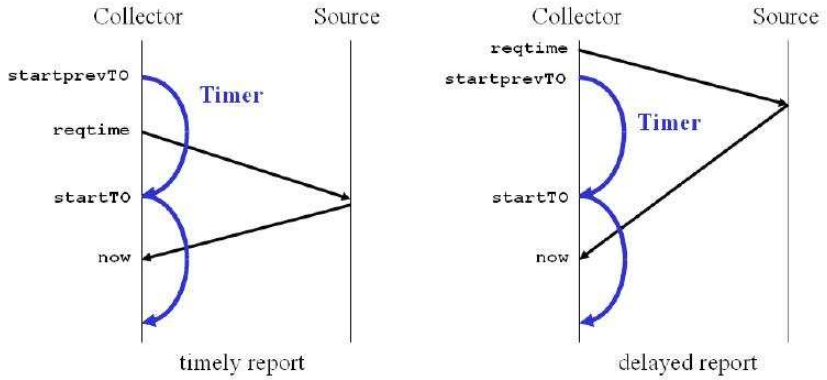

Fig. 1. The two types of reports

Fig. 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.

3 Clustering of information sources

The congestion control mechanism of TICP 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 TICP. First, we detail the reasons that motivated the addition of the clustering mechanism. Then, we describe briefly the GNP Internet coordinate system[6] that provides the knowledge of sources locations and we explain our approach to cluster information sources. Finally, we validate TICP with clustering by the means of simulations and real experiments.

3.1 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.

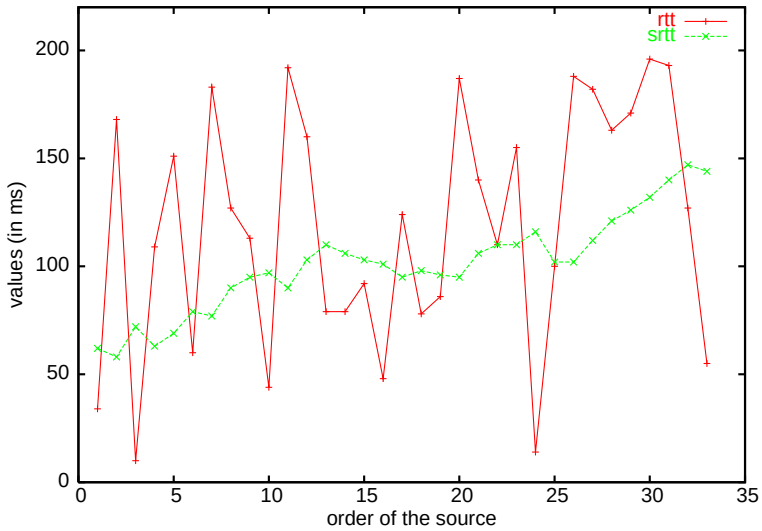

Fig. 2. Estimating RTT with random ordering of sources

To clarify this point, we plot in Fig. 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 simulations whose setting is to be described later. The figure 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

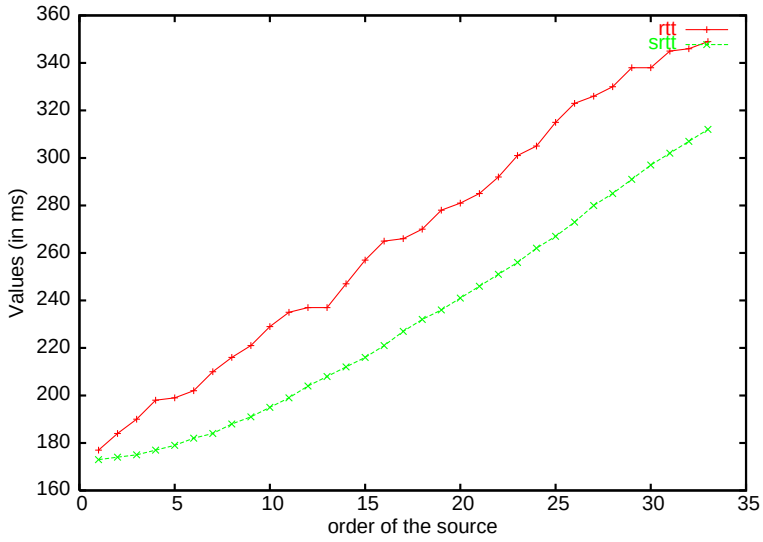

Fig. 3. Estimating RTT with latency-based ordering of sources

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. This way, the RTT in the network varies smoothly and can be estimated more accurately. Fig. 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 smooth shift that can be easily compensated by the mean deviation *rttvar*.

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 to the congestion of several bottlenecks at the same time. But the collector cannot with one control adapt the sending rate of probes to network conditions on several paths in parallel. In case of random ordering, the Internet is viewed by TICP as a sole bottleneck which is far from reality. TICP should treat the bottlenecks one by one. The gathering of information sources into clusters and their probing from the nearest to the collector to the farthest ensures this decoupling of network bottlenecks and adapts correctly the probing rate to these bottlenecks one after the other instead of having a joint adaptation.

3.2 Global Network Positioning System (GNP)

To support the clustering mechanism, we need the knowledge of hosts' locations. Global Network Positioning (GNP) [6] 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 measurements of delay between 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 measurements of delay between itself and the landmarks. We suppose these coordinates are available to the TICP collector in the way the IP addresses of hosts and their names are. Finally, coordinates are meant to form a part of the ID of a machine in the Internet helping to localize it. The way to get this information is out of scope of TICP. One can do it using an additional channel as the DNS or other at a frequency much lower than the frequency of collecting information.

3.3 The clustering approach

A cluster is a set of hosts located in the same neighbourhood such that the latency to reach a host in the same cluster is generally shorter than the latency to reach a host not belonging to it. If the cluster is well chosen, its hosts very likely meet the same network conditions when they communicate with the collector and their traffic cross the same network bottleneck. This is the basic idea behind our solution to remove (even partially) the bias caused by random probing in TICP. To locate information sources in the Internet, we use GNP as described above. We model the Internet as a two-dimensional Euclidean space. The collector and sources participate to GNP as ordinary hosts in addition to some well-defined landmarks. An information source S has a couple of coordinates (x_S, y_S). The collector's coordinates are (x_{co}, y_{co}).

We define a cluster as being a set of information sources whose representing GNP points are located in a square area. The side length of the square is denoted by a, which is a parameter of the protocol. The central cluster is the square whose centre is the point representing the collector.

A cluster is completely defined by a couple of coordinates (X,Y) being integer values. These coordinates are those of the centre of the corresponding square relatively to the collector coordinates and then normalized by a. An information source S whose coordinates are $S(x_S, y_S)$ belongs to the cluster (X,Y) given by:

- $X = round(\frac{x_S - x_{co}}{a})$
- $Y = round(\frac{y_S - y_{co}}{a})$

TICP probes information sources from the nearest to the collector to the farthest one. This is supposed to smooth traffic transitions between network

bottlenecks while maximizing the volume of collected information at the beginning of the session. The collector begins with the central cluster and traverses the other clusters following a spiral trajectory. Fig. 4 illustrates this trajectory. One can with a simple procedure, find the coordinates of the next cluster during the collection knowing the coordinates of the current cluster.

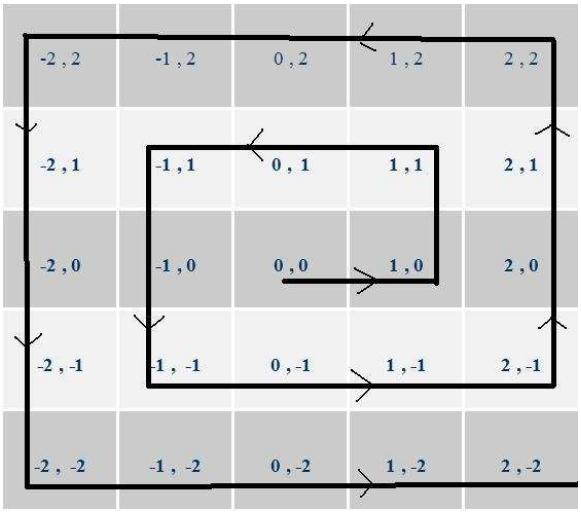

Fig. 4. Order of probing information sources

3.4 Performance evaluation

To assess the impact of clustering sources on the performance of TICP, we run simulations in ns-2 [7] over realistic network topologies as well as real experiments on the PlanetLab testbed using our C++ implementation of the protocol. We are particularly interested in the evaluation of the collection session duration and in the understanding of the impact of the different protocol parameters.

3.4.1 Simulations results

We discuss here the results of our simulations. These simulations are run in ns-2 [7] after the implementation of both TICP and GNP. We generate realistic network topologies for simulations using the GT-ITM tool (Georgia Tech-Internet Topology Modeling) [11]. We choose to work on transit-stub (TS) topologies due to their ability to capture the complexity and the hierarchical structure of the real Internet. TS topologies model the Internet using a 2-level hierarchy of routing domains with transit domains interconnecting lower level stub domains. We assign latencies of 35ms for intra-transit domain links, 10 ms for stub-transit links and 5ms for intra-stub domain links. Fig. 5 shows an example of a TS topology. The size of each topology realization is depicted by

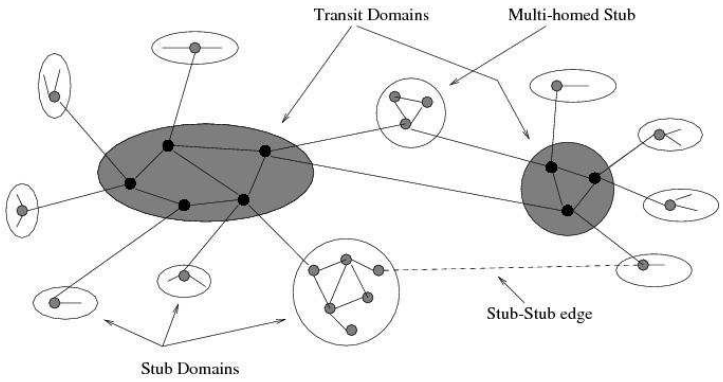

Fig. 5. Transit-stub network topology

the parameter values in Table 1. 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 2. 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 session. Routing tables are calculated at the beginning of simulations to provide shortest paths in terms of number of hops. These tables are used to carry probes and reports in unicast between the collector and the information sources. In contrary to [3] and to make our evaluation of TICP more realistic, multicast routing is disabled in routers.

Table 1
Transit-stub topology 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

We compare between the both versions of TICP with and without clustering of information sources. The comparison criterion is the collection session duration. Fig. 6 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

Table 2
TICP parameters

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

clustering.

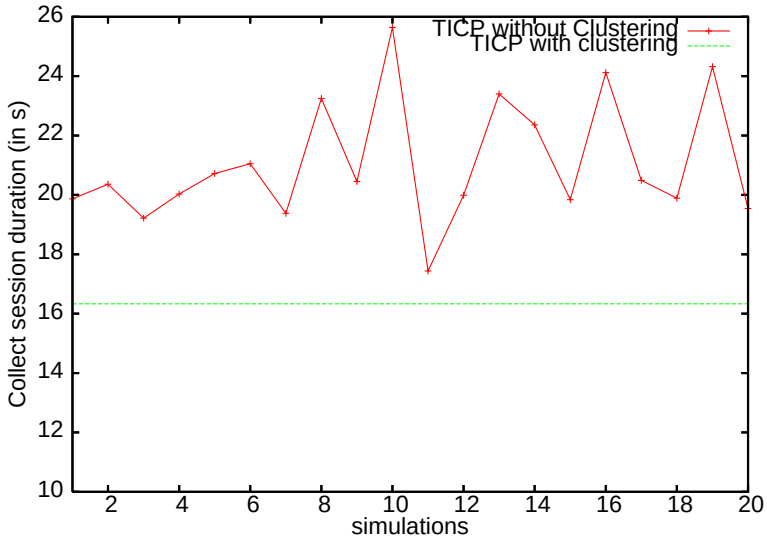

Fig. 6. Collection session duration for different ordering of sources

To evaluate the optimality of TICP, we implement in ns-2 an information collection protocol having a constant window size. For each window size, we run 10 simulations and we record the minimum of the collection session duration over them in order to identify the best combination. Figure 7 plots three curves in the same space. The x-axis represents cwnd, the size of the congestion window for the constant congestion window protocol. The y-axis represents the collection session duration (in seconds) for the constant window protocol. This is the curve having 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. As for the two horizontal lines, they represent the session duration obtained by TICP and are only drawn for comparison. TICP adapts dynamically cwnd to network conditions. The role of TICP is to find in a dichotomic way the right congestion window size that minimizes the collection session duration. This proves the out-performance of an adaptive solution compared to a non adaptive one. We notice in the figure how TICP with clustering manages to reach the optimum unlike TICP without clustering which yields longer durations.

Then, we vary the cluster size and we study its impact on the collection

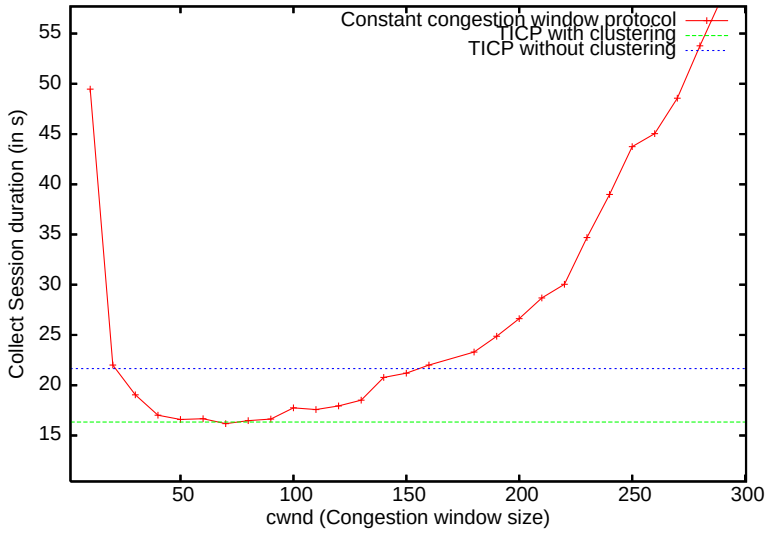

Fig. 7. Optimality of the protocol

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. Fig. 8 validates this intuition where we can see that over the network topologies we simulate 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 1. The number of sources is taken equal to 500.

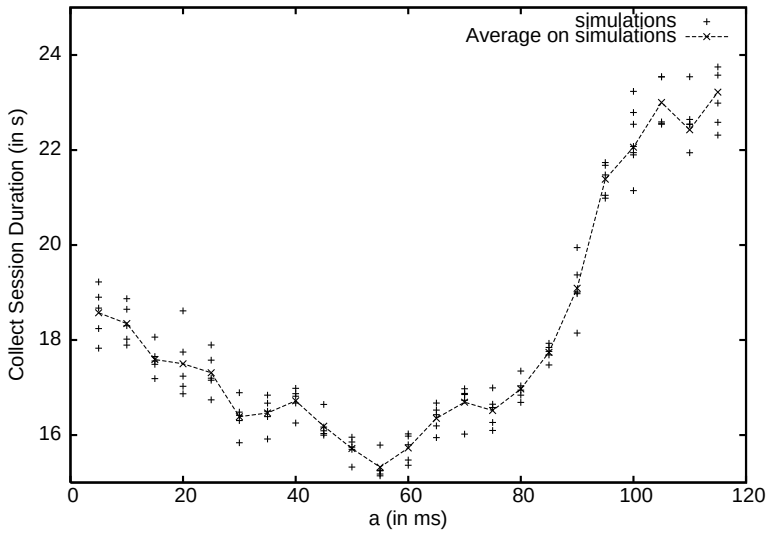

Fig. 8. Impact of a on collection session duration

Fig. 9 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 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 dependent of the topology. As mentioned above, a cluster is in somehow the set of sources located behind the same bottleneck. Thus, increasing the number of sources (distributed uniformly in space) does not increase the number of bottlenecks in the network. Once we reach the maximum number of bottlenecks, the optimal size of the cluster will become constant. One can safely use this value for applications collecting data from a very large number of sources. We suggest that in practice, one starts by calculating this value by running multiple collection sessions, then adapts it as a function of the measured session duration to account for any change in the underlying network topology.

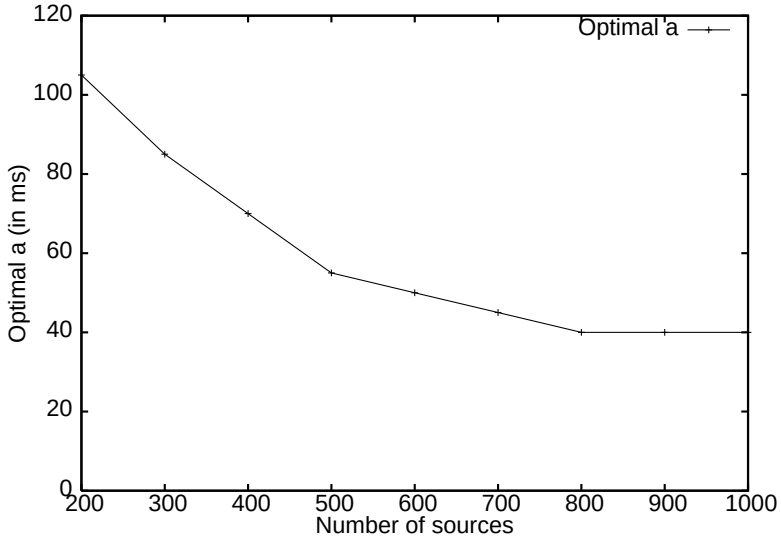

Fig. 9. Optimal a as a function of the number of sources

3.4.2 Experimental results

We implement TICP with clustering in C++ and run real experiments on the PlanetLab testbed [9]. 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 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 of our results. Fig. 10 plots the collection session duration as a function of the number of sources for the optimal cluster size, a small cluster size (100 ms) and a big cluster size (1000 ms); and Fig. 11 plots the

optimal cluster size as a function of the number of sources.

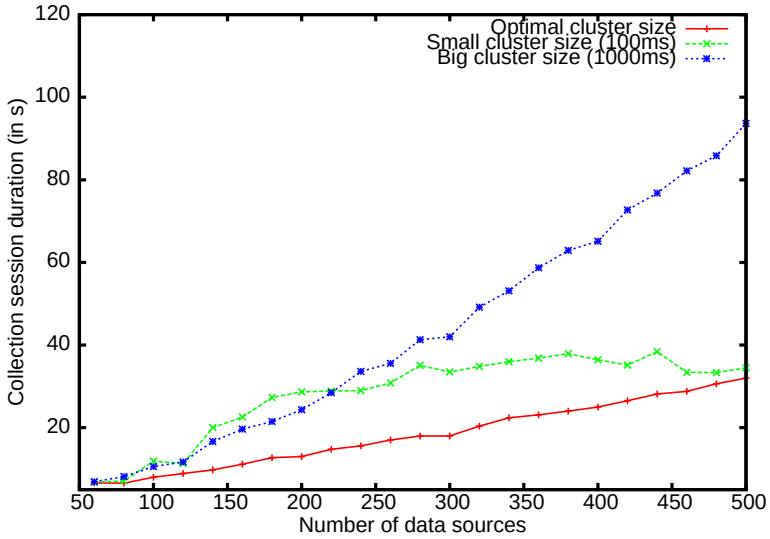

Fig. 10. Collection session duration as a function of number of sources in PlanetLab experiments

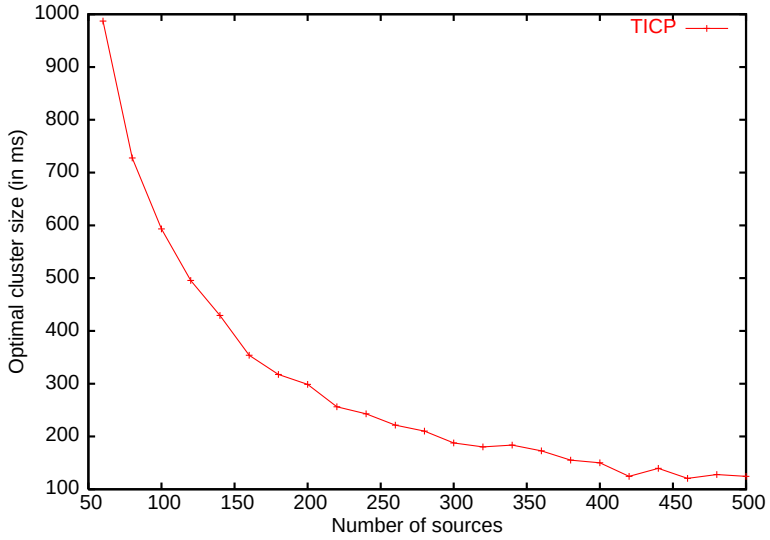

Fig. 11. Optimal cluster size as a function of the number of sources in PlanetLab experiments

Both figures show that experimental results have the same shape compared to simulation ones but clearly different values of the optimums. This is because the real Internet topology is different from the topologies used in our simulations. The main observation we can make from Fig. 10 is that the collection session duration grows almost linearly with the number of sources which means that TICP scales well in practice. Since the large cluster curve corresponds almost to the no clustering case, one can notice that there is an important gain when using TICP with clustering. Moreover, the small cluster curve shows

that having a few number of sources in each cluster is suboptimal. As for Fig. 11, 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.

4 Collecting several packets of data from each source

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 generate data later during the collection session. In this section, we present our extension to TICP that allows collecting informational reports of several packets. We explain first the new functionalities added to TICP. We then study 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.

4.1 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 the 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 set to 1, the collector understands that the source has more packets to send and that it should keep probing it for more packets.

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. 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 discovers that some packets requested during the first round have arrived to the collector while others have been delayed or lost. The probing scheme consists in asking 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 lost to retransmit it. As for sources whose first packet has not yet arrived (their deadlines have not expired), an efficient solution is 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 behavior is generalized to the following rounds.

To implement this behavior, 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]*

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 (*REQUESTS_LIST*) a request packet whose deadline has been reached. We recall that the deadline of a request/data packet is reached when the corresponding request has been transmitted before *startprevTO*, the time at which the previous timer has been scheduled (see Section 2). 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_LIST* and *LAST_SENT_REQUEST* must be incremented. In the case of a retransmitted request, a request packet having the same *RSEQ* is sent and the transmission time 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 2), then it deletes the corresponding request from *REQUESTS_LIST*.

4.3 Scalability of the new version of TICP

We want to test the scalability and the efficiency of this new version of TICP 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. We fix the number of sources and study the impact of changing the number of packets per source on the performance of TICP. Fig. 12 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

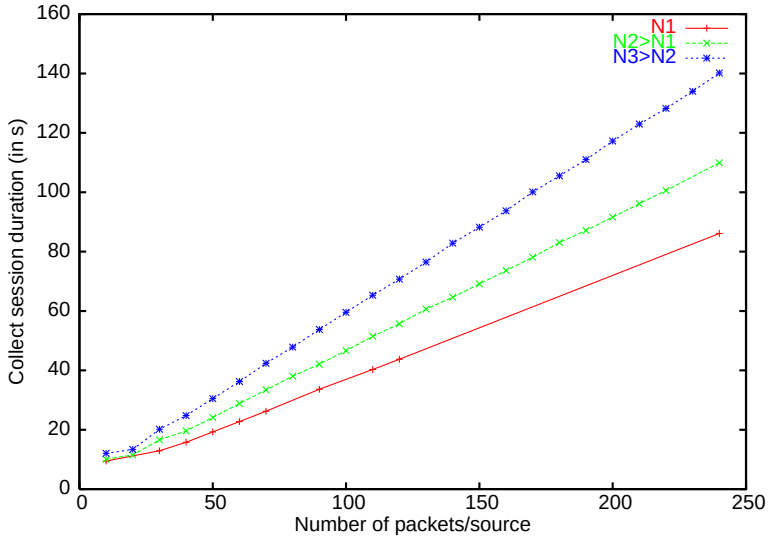

Fig. 12. Impact of the number of packets per source

given number of sources, the collection session duration varies linearly with the number of sources. When the number of sources increases, the slope of the

graph remains constant. This evolution proves the scalability of TICP when the number of packets per source increases. One can interpret this behavior by the fact that the network is seen by TICP 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.

4.4 Comparison with a collection application using parallel TCP connections

The intuitive question that one would ask at this level is how TICP compares to a collection application using regular TCP to collect data from sources (one TCP connection per source). As we are in the context of relatively small number of packets from each data source, the TCP connections will be short-lived connections, and hence, TICP will provide better performances for the simple reason that it avoids the three-way handshaking and the slow start phases of the individual TCP connections. Note that for very large amounts of data per source, TICP works as well but its gain compared to the use of TCP is not guaranteed. It is still has the advantage of doing the collection in one session for all sources without the need to establish and schedule a large number of TCP connections.

We compare here TICP 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 lets 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 minimizes 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. 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 deteriorates. 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. We believe this should provide a fair comparison with between TCP and TICP.

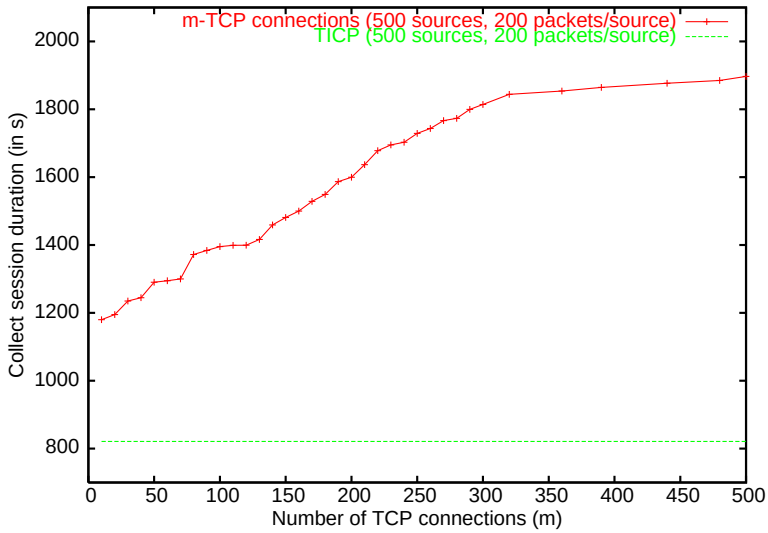

Fig. 13. Parallel TCP connections vs. TICP

We implement this parallel TCP based 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 TICP under this configuration. Then, we run the collection application for different values of the number of parallel TCP connections m. Fig. 13 plots two curves in the same space. The first curve is for the TCP based collection application and the second one is for TICP and is present only for comparison. In the figure, the x-axis represents the number of parallel TCP connections (m) for the collection application using TCP. The y-axis represents the collection session duration. As TICP opens one connection to all sources, the duration of its session is independent of the value of m on the x-axis. As expected, we can see 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 TICP is shorter than the collection session duration for any number of parallel short-lived TCP connections. This proves the efficiency and the importance of TICP for the collection of several packets 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 TICP, 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 TICP 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.

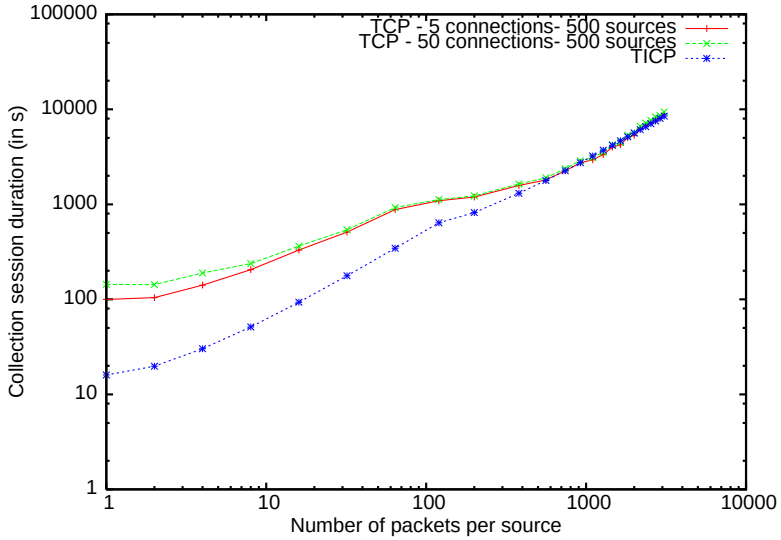

Fig. 14. Impact of the number of packets per source

To study the impact of the number of packets on the performance of both TICP and parallel TCP connections, we do further simulations varying the number of packets per source. For the TCP-based collection, we consider the two cases of 5 and 50 connections. Figure 14 plots on a log-log scale the collection session duration as a function of the number of packets for TICP, TCP-5 connections and TCP-50 connections. As expected, this figure shows the outperformance of TICP when the number of packets to collect is relatively small (from 1 to 500 packets here). We can notice a gain of one order of magnitude due to avoiding the slow starts phases of the individual TCP connections. For large files however, the gain reduces and both TICP and parallel TCP perform equally well. This is because large files are able to bring the individual TCP connections to their stationary regimes, hence reducing the gain introduced by the traffic multiplexing in TICP. For small files this stationary regime is barely reached making the slow start phases dominant and network ressources under utilized if TICP multiplexing across sources is not used.

5 Parallelizing the collection using proxy collectors

Even though TICP 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. 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 extend TICP to support delegation. Finally, we discuss the simulation results.

Note that in some cases the delegation of collection can influence the collected data itself. This can be seen for example when part of the collected information is related to the characteristics of the path between the collector and the source of information as the delay and the bandwidth. The delegation in this case should be done with further attention and its impact on the data itself is to be studied carefully before deploying proxies. These specific cases are out of the scope of this work. We only deal here with the network load and we suppose that the collected information is the same whether it is probed from machine A or machine B. Our optimization only aims to reduce TICP traffic and fastens the collection while collecting the entire data. The specific cases where the data to collect is function of the way the collection is done is an important problem to consider but is somehow application specific, so to stay general we leave it for a future research.

5.1 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 x F-> 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 [10]. 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 solutions in short running time.

5.2 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, it plays the role of a regular TICP collector towards ordinary sources in its charge. This behavior is described with details in Section 4.

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 and that the proxy collector is waiting for new packets to arrive from ordinary sources. In this case, the proxy collector replies with a *persistency* 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 persistency packet in order for the main collector not to consider the absence of a response as an indication of network congestion. Such a persistency packet has the same sequence number as the requested packet but with a *MORE* field set to 2.

We run simulations in ns-2 over network topologies described in Section 3 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. Fig. 15 plots the results for 500 randomly distributed sources, with the x-axis showing the number of proxy collectors. 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 an 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. For the extreme case where the number of proxy collectors is equal to the total number of sources, the proxy collectors have no ordinary sources in their responsibilities, so the collection operation is similar to an ordinary TICP collector gathering data from all sources. Generally speaking, delegating to everyone is equivalent to not delegating at all. The collect session duration for 500 proxy collectors (the point on the very right hand side of the figure) is equal to that for zero proxy collector (the point on the very left hand side of the figure).

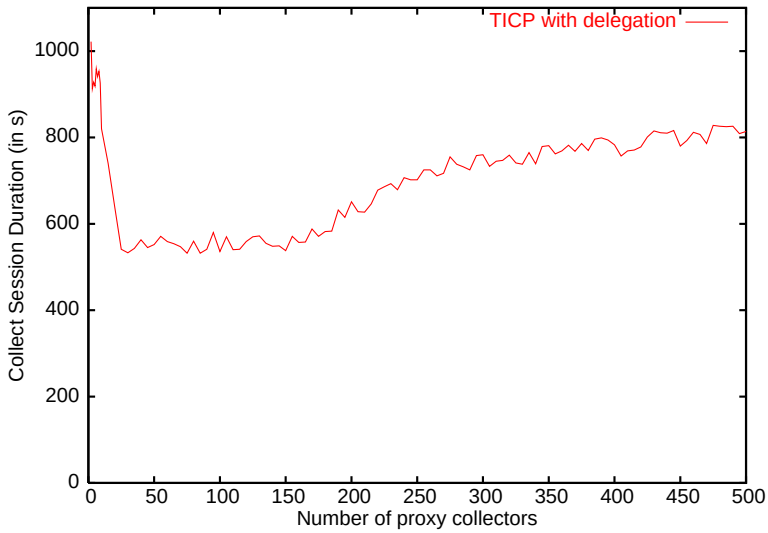

Fig. 15. TICP with delegation for 500 sources

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 Fig. 17. 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).This observation is illustrated in Fig. 16 where we plot the collection session duration as a function of the number of proxies for 100 sources randomly distributed in the network. As the number of sources is less than the threshold, the performance keeps improving by adding more proxy collectors. In this case, the population is small and one TICP session is able to do a good

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 a 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 maximum number of delegated proxies to be used is topology dependent. Our future research will focus on its calculation for different scenarios.

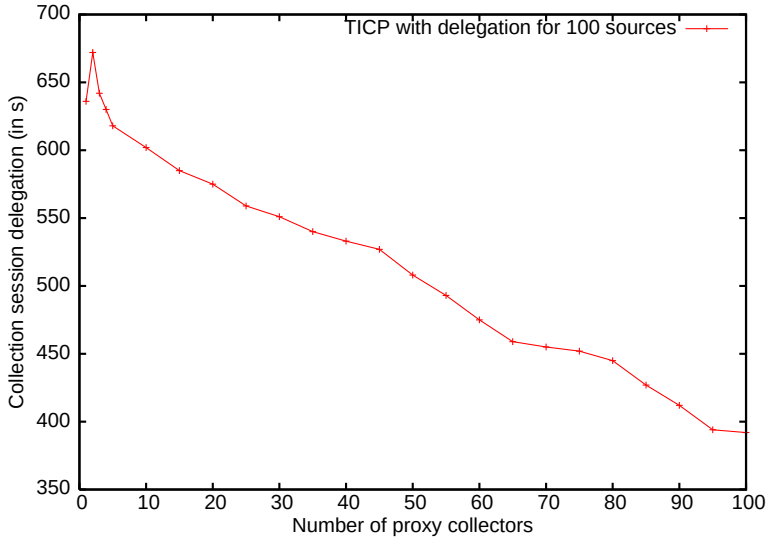

Fig. 16. TICP with delegation for 100 sources

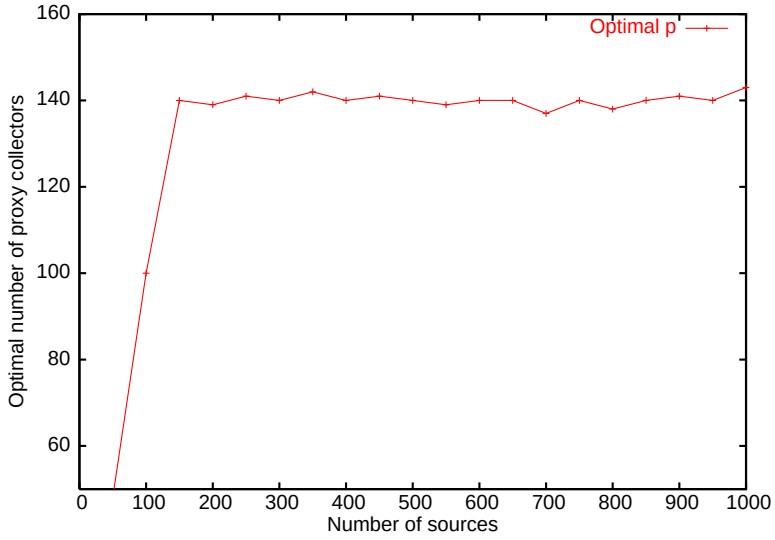

Fig. 17. Optimal number of proxy collectors

6 Related work

There is actually no general stand-alone transport solution to collect information from a large number of network entities on an end-to-end basis while providing congestion and error control. Most if not all existing solutions are application specific that do not answer all the questions we rise in this work. Some solutions provide partial collection because the applications don't need it, others require the collaboration of intermediate nodes and routers and so are not general, and there are some solutions close to ours but use simple congestion control as periodic or exponential probing at a low rate in order not to congest the network.

TCP is a protocol to collect one file from one machine. Ensemble-TCP [20] and HTTP 1.1 [21] are two protocols to collect many files from one machine using one single TCP connection. SNMP [15] and measurement infrastructures as Skitter [17] implement periodic and exponential probing at a low rate to collect information from routers and to probe machines for the purpose of delay and topology measurements. When it comes to data disseminated through the Internet, there is no clear solution to achieve this collection on an end-to-end basis. Though some protocols like Concast [14] use the principle of active networks to program routers so that they can participate to the collection.

There has been some effort in the literature to develop collection solutions for reliable multicast applications. Indeed, in reliable multicast, sources that did not receive a packet need to send a NACK asking the collector for a retransmission of this packet (we keep the terms collector and sources even though the collector in this context is transmitting data and the sources are listening). Sending many NACKs may cause congestion in the network or at the collector. The problem is called NACK implosion and several solutions have been proposed in the literature to collect these NACKs while avoiding congestion. The difference from our context is that the NACK information can be safely filtered; there is no need that a host sends a NACK if another host has already sent a NACK for the same packet. Thus, it was proposed to aggregate NACKs either along a tree that connects all sources or using multicast itself. In [12], 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 [13,16], 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 [14].

The reliable collection of information has also its application in sensor net-

works. 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., [18], [19]. These protocols agree on that end-to-end transport solutions lead to poor performance in this case 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 TICP provides.

7 Conclusions and perspectives

TICP 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 explains and validates three main components of TICP. First, to ensure a smooth variation of the congestion control parameters, we add to TICP a mechanism to cluster information sources. Simulation and experimental results show that this mechanism ameliorates the performance by reducing the loss ratio of reports, and therefore shortens the collection sessions. Second, we extend the protocol to support collecting several packets of data from each source. We prove by simulations that our approach is scalable and that TICP performs better than a collection application using parallel TCP connections. Lastly, we enhance the collection further by delegating it to some intermediate nodes called proxy collectors. We modify TICP to support this delegation and to choose optimally the proxy machines. The simulations showed that for a well-chosen number of proxy collectors, TICP 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 TICP in the cases of several packets of data per source and delegated proxies. We are also working towards the integration of TICP in Internet measurement infrastructures for the purpose of controlling the rate of probing packets and for the collection of passive measurements carried out at different links inside the network.

References

[1] M. Allman, H. Kruse, S. Ostermann, "An Application-Level Solution to TCP's Satellite Inefficiencies", Proceedings of the First International Workshop on Satellite-based Information Services (WOSBIS), New York, Nov. 1996.

[2] M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control", RFC 2581, Apr. 1999.

[3] C. Barakat, M. Malli, N. Nonaka, "TICP: Transport Information Collection Protocol", in Annals of Telecommunications, vol. 61, no. 1-2, pp. 167-192, Jan. 2006.

[4] Mohamed Karim Sbai, Chadi Barakat, "Transport Information Collection Protocol with clustering of information sources", in proceedings of NTMS 2007 (Conference on New Technologies, Mobility and Security), Paris, May 2007.

[5] 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), Dec. 2006, vol. 24, no. 12.

[6] T. S. Eugene Ng and H. Zhang, "Towards Global Network Positioning", ACM SIGCOMM Internet Measurement Workshop 2001, San Francisco, Nov. 2001.

[7] The Network Simulator ns-2, http://www.isi.edu/nsnam/ns/

[8] V. Paxson, M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, Nov. 2000.

[9] The PlanetLab platform, http://www.planet-lab.org/

[10] M. G. C. Resende and R. F. Werneck, "A hybrid heuristic for the p-median problem", Journal of Heuristics, vol. 10, pp. 59-88, 2004.

[11] E. W. Zegura, K. Calvert and S. Bhattacharjee, "How to Model an Internetwork", IEEE Infocom '96, San Francisco.

[12] Lin (J.C.), Paul (S.), RMTP: A Reliable Multicast Transport Protocol.IEEE INFOCOM, March 1996.

[13] 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.

[14] 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.

[15] Case, J., M. Fedor, M. Schoffstall, and J. Davin. 1990. A Simple Network Management Protocol (SNMP). RFC 1157, May.

[16] Lacher, M. S., Nonnenmacher, J., and Biersack, E. W. 2000. Performance comparison of centralized versus distributed error recovery for reliable multicast. IEEE/ACM Transactions on Networking. 8, 2 (Apr. 2000), 224-238.

[17] Skitter infrastructure by CAIDA, http://www.caida.org/tools/measurement/skitter/

[18] Stann (F.), Heidemann (J.), RMST: Reliable Data Transport in Sensor Networks. IEEE International Workshop on Sensor Net Protocols and Applications (SNPA), May 2003.

[19] Wan (C.), Campbell (A.), Krishnahmurthy (L.), PSFQ: A Reliable Transport Mechanism for Wireless Sensor Networks. ACM International Workshop on Wireless Sensor Networks and Applications, September 2002.

[20] Eggert, L., Heidemann, J., and Touch, J., "Effects of Ensemble TCP," ACM Computer Comm. Review, January 2000.

[21] RFC 2616, Hypertext Transfer Protocol http://tools.ietf.org/html/rfc2616