Transport-Layer Issues in Information Centric Networks

Stefano Salsano, Andrea Detti, Matteo Cancellieri, Matteo Pomposini, Nicola Blefari-Melazzi
Department of Electronic Engineering, University of Rome "Tor Vergata" / CNIT Research Unit
Via del Politecnico 1, Rome (Italy)

{stefano.salsano, andrea.detti, matteo.pomposini, blefari}@uniroma2.it

ABSTRACT
Content to be transported over an Information Centric Networking (ICN) infrastructure can be very variable in size, from few bytes to hundreds of gigabytes. Therefore it needs to be segmented in smaller size data units, typically called chunks, in order to be handled by ICN nodes. A chunk is the basic data unit to which caching and security (e.g. encryption and signature) functions are applied. If we consider the overhead and the number of cryptographic operations to be performed by nodes, a good choice for the chunk size would be from hundreds of KBs up to few MBs. However, if the chunk size is bigger than the Maximum Transfer Unit of a link, chunks will be fragmented. We show that if we have more than 3-4 fragments per chunk, and congestion and reliability functions are executed on a chunk by chunk basis, the efficiency of the congestion control algorithm drastically decreases. On the other side, a small chunk size would increase overhead and rate of signature checks.

The contribution of this paper is twofold: 1) we propose to segment content in two levels: at the first level the content is segmented in chunks, at the second level the chunks are segmented into smaller data units, handled by an ICN specific Transport Protocol (ICTP), performing reliability and congestion control functions; 2) we propose to adopt a receiver-driven transport protocol, in which the receiver adjusts the sending rate to control congestion, we describe an implementation of this protocol, and evaluate its performance.

Categories and Subject Descriptors
C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords
Internet architecture, Information Centric Networking, Transport protocol, Chunk, Fragmentation, Test-bed, Performance.

1. INTRODUCTION
Information Centric Networking (ICN) is a concept proposed some time ago under different names [1][2], which is attracting more and more interest, recently (see e.g. the papers [3][4][5]). The basic functions of an ICN infrastructure are to:

- address contents, adopting an addressing scheme based on names (identifiers), which do not include references to their location;
- route a user request, which includes a "destination" content-name, toward the "closest" copy of the content with such a name;
- deliver the content back to the requesting host.

In our view, an ICN would offer the following advantages:
i) efficient content-routing. ii) in-network caching; iii) simplified support for peer-to-peer like communications, without the need of overlay dedicated systems; iv) per-content quality of service differentiation, providing different performance in terms of both transmission and caching; v) handling of mobile and multicast communications, simplifying handovers and stateful nodes; vi) content-oriented security model. Securing the content itself, instead of securing the communications [4]; vii) support for time/space-decoupled model of communications, providing publish/subscribe service models and allowing "pieces" of network, or sets of devices to operate even when disconnected from the main Internet (e.g. sensors networks, ad-hoc networks, vehicle networks, delay-tolerant-networks, social gatherings, mobile networks on board vehicles, trains, planes).
In the CONVERGENCE project [6], we identified eight fundamental components of an ICN infrastructure, which need to be addressed: i) primitives & interfaces; ii) naming scheme; iii) forward-by-name (or route-by-name) mechanism, used by ICN nodes to relay an incoming content request to an output interface; iv) the routing protocols used to disseminate information about location of contents, so as to properly setup the name-based forwarding tables; v) the data forwarding mechanism that allows the content to be sent back to the device that issued a content request; vi) in-network caching; vii) segmentation & transport mechanisms; viii) security & privacy.
In this paper we focus on the segmentation & transport component. We first recall an architecture that we proposed in the CONVERGENCE project [6] and in [7] and then we propose our main contribution, which is a segmentation scheme and a receiver-driven transport protocol.

2. THE CONET ARCHITECTURE
In [7] we proposed an architecture called CONET (COntent NETwork), which tries to achieve the pros of ICN, while mitigating the cons. A CONET is defined as inter-network that connects CONET Sub Systems (CSSs) (see Figure 1). A CSS contains CONET nodes and exploits an under-CONET technology to transfer data among CONET nodes. The devices within a CSS can use an autonomous and homogeneous under-CONET addressing space and an interior under-CONET routing protocol. A CSS could be:

1. a couple of nodes connected by a point-to-point or an overlay link (CSS n.1 of Figure 1);
2. a layer 2 network like Ethernet (CSS n.3 of Figure 1);
3. a layer 3 network, e.g. a private IPv4/IPv6 network or a IPv4/IPv6 subnet or a whole Autonomous System or even the whole current Internet (CSS n.2 of Figure 1).

ICN'12, August 17, 2012, Helsinki, Finland.

CONET nodes exchange CONET Information Units (CIUs): *interest* CIUs convey requests of named-data; *named-data* CIUs transport chunks of named-data, e.g., parts of a file. To best fit the transfer units of an under-CONET technology, all CIUs are carried in smaller CONET data units named carrier-packets.

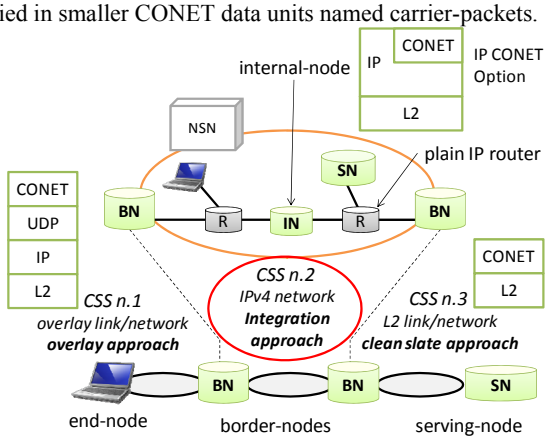

Figure 1. CONET Architecture

CONET nodes are classified as end-nodes (ENs), serving-nodes (SNs), border-nodes (BNs), internal-nodes (INs) and name-routing-system nodes (NRSs). End-nodes are user devices that request named-data by issuing interest CIUs. Serving-nodes store, advertise and provide named-data by splitting the related sequence of bytes in one or more named-data CIUs, which are transferred by means of carrier-packets. Border-nodes, located at the border between CSSs, forward carrier-packets by using CONET routing mechanisms (i.e. taking into account the requested content-name: routing-by-name) and cache named-data CIUs. Optional Internal-Nodes could be deployed inside a CSS to provide in-network caches; differently from border-nodes, internal-nodes forward carrier-packets by using only under-CONET routing mechanisms. Name-Routing-System nodes are used in a CSS to assist the CONET routing-by-name process.

As shown in Figure 1, Border Nodes interconnect different CSSs, therefore the end-to-end forward-by-name process can be seen as the process of finding a sequence of Border Nodes from an End-Node up to a Serving Node. In short, CONET is an interworking protocol, just like IP.

The operation in a CONET internetwork can be described as follows. A Border Node checks if the content requested is available in its cache; if not it performs forward-by-name. If the CSS is an IP network, the result of the forward-by-name operation is the IP address of the upstream Border Node, therefore the content request can be sent using this destination IP address. An Internal Node in the path between the two border nodes "intercepts" the content request, it checks if the requested content is available in its cache, if not it forwards the packet using the IP destination address. A plain IP Router in the path between the two Border Nodes will simply forward the packet looking at the IP destination address. When data packets providing the requested content are generated by the Serving Node towards the End-node (or by any Border or Intermediate Node that had cached the content), the crossed downstream Border Nodes and Internal Node can in turn cache the content while forwarding it. In this way, further content requests for the same content will not need to travel up to the Server Node.

Note that the three typologies of CSSs depicted in Figure 1 correspond to different deployment scenarios, respectively:

- overlay: CONET on top of the IP layer, as it occurs in the CSS n.1 of Figure 1;
- clean slate: CONET on top of layer-2, as it occurs in the CSS n.3 of Figure 1;
- integration: CONET integrated in the IP layer, as it occurs in the CSS n.2 of Figure 1.

The first two approaches are known in the literature. The integration approach supports CONET in a CSS that is an IP network (IP-CSS). Depending on where CONET routing protocols are deployed (i.e. where we deploy Border Nodes) we can have different scenarios: if CONET protocols are implemented only in user equipments, interconnected by the current Internet, then we have only one CSS: the current Internet. If they are implemented in current border gateways (i.e. where BGP typically runs), then CSSs coincide with current Autonomous Systems. If they are implemented in all current routers, then CSSs coincide with current IP subnets.

Additionally (but optionally), we propose to make IP itself content-aware, by transporting the identifier (name) of a CONET carrier-packet in a novel IPv4 or IPv6 option, which we name CONET option [8]. The advantages of this approach with respect to the overlay one is that it allows nodes to quickly forward carrier-packets, without the need of terminating the UDP protocol and passing through the application layer or performing a "deep packet inspection". In addition, this approach allows deploying CONET routing-by-name functions only in a subset of nodes (i.e. Border-nodes and End-nodes) while allowing caching in all nodes running the new IP option (i.e. Internal nodes).

2.1 Model of operations

To provide a content, a server splits the content in blocks of data, named *chunks*, and assigns a unique network identifier to each chunk. A network identifier is a string like "cnn.com/text1.txt/chunk1", which is said to be the "name" of the chunk. The role of the ICN protocols is to discover and deliver named chunks. In order to fetch a chunk, a user issues a data unit, named *interest CIU* or simply *Interest*, that contains the name of the chunk. ICN nodes *route-by-name* the Interest, by using a longest prefix matching forwarding strategy and a name-based routing table. We name the entries of the name-based routing table as *ICN routes*. An ICN route has a format like <name-prefix, output port identifier, next hop information>. A name-prefix should be either the full name of a chunk, e.g. "cnn.com/text1.txt/chunk1", or a continuous part of it, starting from the first left character e.g. "cnn.com/".

The first en-route device that has the chunk sends it back in a named-data CIU, or simply *Data*. Network nodes forward Data towards the client, through the same sequence of ICN nodes previously traversed by the Interest message. This can be accomplished in two ways. The Data forwarding process can exploit reverse-path information contained in the header of Data message, as previously collected in the Interest message during its forwarding process (see reverse-path source-routing in [9]). An alternative is to exploit reverse-path information temporarily stored in the traversed nodes during the Interest forwarding process (see Pending Interest Table of [3]). In either cases, the routing-by-name process does not involve Data messages, but only Interest messages. Downloading a whole content is achieved by sending a *flow* of Interest messages to retrieve all the chunks of the content. We assume that the sending rate of Interest messages is regulated by a receiver-centric congestion control mechanism [10], as suggested also in [3].

2.2 Naming

As regards the naming scheme, several proposals (e.g. [2][3][4] [9]) agree in adopting a hierarchical naming. Here we assume a rather general hierarchical naming scheme where a name is formed by a sequence of *Components*: a name has the form "Component_1/Component_2/../Component_n". This scheme supports current Web URL, where Component_1 is the domain name (e.g., "cnn.com") and next Components represent the path of the local resource (e.g., /text1.txt). In addition to these Components, which represent the *content-name*, ICN requires other specific Components, e.g. to represent the chunk number ("/chunk1"), version, etc. A hierarchical naming is also able to support human readable names [3][4] and self-certifying names [2] [9] (where Component_1 is the *Principal* and Component_2 is the *Label*). The full sequence of Components is referred to as the *chunk-name*.

We conclude this section by remarking that our architecture is similar to CCN [3], from which it differs in the following aspects:

- we designed an *interworking* architecture, allowing for different deploying alternatives (clean slate, overlay, integration);
- we proposed the integration approach, which consists in extending the IP protocol with new header options;
- we proposed an alternative for data forwarding, which exploits reverse-path information contained in the header of Data message, instead of information temporarily leaved in the traversed nodes;
- we specified a routing scheme (in another paper submitted for publication and in [14]);
- we are proposing (in this paper) a different segmentation strategy and specifying a transport-layer protocol.

We will now focus on segmentation and transport issues; our proposed solutions can be of course applied to our own architecture but also in other ICN systems that route-by-name content requests such as CCN [3] [4].

3. SEGMENTATION AND GENERAL ISSUES

Content to be transported over an ICN can be very variable in size, from few bytes to hundreds of Gigabytes (and it is not easy to set an upper bound to content size). Therefore content needs to be segmented in smaller size data units, typically called *chunks*, in order to be handled by the ICN nodes. A chunk is the basic data unit to which caching and security is applied. This means that: i) a single chunk out of a larger content can be requested by an End-Node; ii) single chunks will be *signed* by the origin Server-Node for security reasons; iii) Border and Intermediate Nodes will authenticate the chunks and store them in their caches as needed.

Each chunk contains an amount of overhead, mostly due to security information (e.g. the signature needed to authenticate the chunk) which is more or less independent from the chunk size. Taking the CCNx [12] implementation as reference, this overhead is in the order of 650 bytes per chunk. Each signature and authentication operation is computationally very expensive and again this cost is mostly independent from the chunk size (this is because the computationally expensive operations, based on asymmetric cryptography, are applied to a fixed size hash of the chunk, while the cost of producing the hash, which depends on the chunk size, can be neglected).

These considerations on the overhead, and on the number of cryptographic verification operations to be performed by caching nodes, suggest the chunk size to be rather large. We argue that reasonable chunk sizes can be in the order of hundreds of KBs (e.g. 128, 256, 512) up to few MBs (e.g. 1, 2, 4). When chunks of such relatively large size are exchanged among ICN nodes, they need to be fragmented, to be handled by level 2 technologies, which normally prescribe a Maximum Transfer Unit of much smaller size. For example CCN [3] uses the chunk as the fundamental transport unit to be exchanged among nodes and relies on the underlying technologies and protocols to actually forward them (e.g., UDP tunnelling in the overlay approach used by the CCNx implementation [12]). In CCNx, if the chunk size is bigger than the MTU, the chunks are subject to IP level fragmentation. Let us denote by *Nc* the number of fragments per each chunk (i.e., *Nc*=Chunk Size / MTU). In case of loss (for example due to congestion) a larger *Nc* has a negative effect on performances (and efficiency), as the loss of a single fragment will lead to the loss of the whole chunk to which the fragment belong.

We observe that, with this approach, the transport mechanism can work at maximum efficiency when the chunk size is smaller than the Maximum Transfer Unit (MTU) over the crossed link layers. In addition, in these conditions, assuming to use a TCP or TCP-like algorithm, it is not possible to achieve fairness. In fact, TCP flows will cause the starvation of ICN flows in case of congestion. For this reason, the default value of chunk size in CCNx [12] implementation is set to the relatively small default value of 4 KB, i.e. roughly 3 IP packets of 1500 bytes. This implies a 10% overhead on the bit rate and the need to perform signature checks at a very high rate, compared to what would have been possible with a chunk size in the order of hundreds of KBs or few MBs, as argued above. Both these phenomena (efficiency and fairness) are further analysed in section 6 below and support our considerations above.

We also note that the reliability and congestion control aspects are handled on a chunk by chunk basis by the CCN applications. In other words, the typical operations performed by a transport protocol are handled at the application level in the CCNx implementation. We argue that it would be useful to introduce a transport protocol below the application level. We propose to handle the segmentation of content in two levels: at the first level, content is segmented in chunks; at the second level, chunks are segmented into smaller data units, handled by an ICN specific transport protocol performing reliability and congestion control, much like TCP does for the transfer of a byte stream in current TCP/IP networks. This approach is represented in Figure 2.

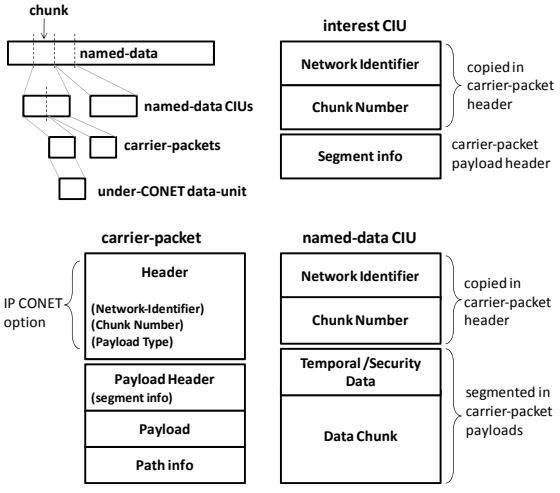

Figure 2. CONET Information Units and carrier-packets

CONET nodes exchange CONET Information Units (CIUs): interest CIUs convey content requests; data CIUs transport chunks of content, e.g., parts of a file. In order to handle the fragmentation in an efficient way, fitting the transfer units of the under-CONET technologies, all CIUs are carried in smaller data units named carrier-packets.

4. TRANSPORT PROTOCOL

We know that TCP endpoints exchange Segment/Ack sequences, the sender sends the Segments and the receiver replies with Acks. The sending rate of Segments is under the control of the sender (which considers the feedback coming from the Acks. We adopt a receiver-driven protocol where endpoints exchange Interest-Data sequences and the exchange rate is regulated by the receiver, following principles put forward in [3] and [10].

This protocol, which we name Information-Centric Transport Protocol (ICTP), implements the same algorithms of TCP (slow-start, congestion avoidance, fast retransmit, fast recovery). It issues a sequence of interest CIUs and each of them requests only a fraction of a data CIU. By controlling the sending rate of these interest CIUs, we obtain a transport protocol that is TCP friendly and provides fairness both among competing ICN flows and among ICN and TCP flows. In [11], the authors also propose a receiver driven approach called ICP (Interest Control Protocol), defining specific congestion control mechanisms and providing a theoretical performance analysis and a validation with packet level simulation. Differently from ICP, we do not introduce radically new congestion control mechanisms, but adapt the well-known TCP ones to the receiver driven context. We do not provide a theoretical analysis, but we have a running implementation interoperable with CCNx applications and we used this implementation for an experimental performance analysis. Note that in the current CCNx [12] implementation the transport protocol (reliability and congestion control) is implemented at the application level, above the API toward the ICN layer (maybe just as a temporary solution). We believe that this is not efficient and we prefer to embed the transport protocol below the API towards the application, just above the networking layer, as in TCP/IP. The details of the ICTP transport protocol in terms of bit and bytes can be found in our internet draft [13], while a more detailed discussion and analysis of performance issues are available in our technical report [14]. Here, we present the main characteristics of the protocol, we describe an implementation of this protocol, and evaluate its performance.

4.1 Protocol description

Rather than designing a new protocol from scratch, ICTP borrows a number of lessons from decade-worth of researches in TCP, adapting them to its receiver driven nature. A TCP-like receiver driven protocol has the advantages of providing:

- *Elasticity*: TCP efficiently uses bandwidth by increasing or decreasing the congestion window, facing decrease or increase of loss/congestion.
- *Fairness*: Coexistence with "legacy" TCP flows without starvation.

We specify ICTP by describing its differences with respect to TCP:

- Interest/Data messages replace TCP Segment/ACK segments
- the sending rate is under the control of the receiver and not of the sender
- the congestion window specifies the Data range requested in Interest messages, rather than how many bytes the sender can transmit
- the equivalent of a duplicate ACK is a hole in the flow of data; data received after a missing part of the expected set of data is considered as a duplicate ACK.

Then, we have some mechanisms which are specific of ICTP:

- *Pre-fetch*: applications can request the download of specific chunks of content, independently of one another. In principle, the transport can request the segments of a given chunk only after it has received the request for such chunk from the application. Applications may implement retransmission and congestion control mechanisms, therefore ICTP could receive requests for more than a chunk in parallel. The interaction between the congestion control performed at application level and the congestion control performed at transport level could prove to be inefficient. We envisage a "prefetch option" allowing retrieving all the (rest of) content. With prefetch, if the ICTP congestion window has space left, bytes for the next expected chunk are requested, without waiting for an explicit request from the application level. Prefetch allows applications to get a content, without having to implement a chunk-based flow control. Without this mechanism, current applications cannot fully exploit ICTP features.
- *Request chunk information*: when the transmission starts, the receiver may not have valid information on chunks. Therefore the requesting node must issues a request for (at least) the chunk size. The corresponding Interest carrier packet is set with a specific flag, and requested Data Carrier Packet will carry at least the chunk size. For the sake of simplicity, this mechanism assumes a fixed chunk size for each content. More complex scenarios are possible, allowing application to publish content with variable chunk sizes (details in our internet draft [13]).

Finally, as regards overhead performance, ICTP overhead is slightly higher than CCN, due to the segmentation overhead to be transported in each fragment (details in our technical report [14]).

5. IMPLEMENTATION

Our implementation is based on CCNx [12] and our modules are designed so as to preserve compatibility with CCN to the greatest possible extent. It is possible to run applications designed for CCNx over our ICTP. Implemented modules are available under a GPL open source license in http://netgroup.uniroma2.it/CONET/.

Figure 3 shows the implemented modules. A modified version of the CCNd daemon included in the CCNx distribution provides caching and security functionality for the chunks. When a requested chunk is not in the cache, the CCND performs the forward-by-name operation by querying the local CONET "lookup-and-cache" module for the next hop.

The lookup-and-cache module implements a route discovery algorithm and a route caching technique, which we designed to improve the scalability of the routing by name functionality. Basically, with this approach, the Forwarding Information Base (FIB) of a node is used as *cache of routes*, while the Routing Information Bases (RIB) is stored in a remote and centralized routing engine, which serves all the node of a CONET Sub-System (CSS) and runs on a centralized server, named *Name Routing System* (NRS) node. If the name to next-hop mapping is not cached in the local table, the lookup-and-cache module queries the NRS node.

When Interest/Data messages need to be exchanged among CONET nodes, the segmentation and transport modules comes into play, replacing the CCNx interest and data packets with

CONET carrier packets. As for the segmentation and transport module we made two implementations: a first one runs in user space and conveys the CONET protocol information in an UDP payload, simply extending the CCNx tunnelling approach; the second one (implementation still ongoing) runs in kernel space and it carries the CONET protocol information in the IP option.
The testbed consists of:

- A PC playing the role of Serving Node, which runs CCNd and hosts the content loaded via the CCN repository.
- A PC playing the role Client Node, which runs CCNd and ccncatchunks2 to retrieve the files.
- A PC playing the role of Border Node which runs CCNd; it forwards interests and data between server and client and it is able to cache content and to serve contents to a client.
- Clients, Border Node and Server Node run a version of CCNx software, release 0.5.0 [12], modified to include our Lookup-and-Cache routing scheme.
- All devices run the Linux OS.

Both CCNx and CONET configurations have been fine-tuned to achieve best performance (for details see [14]).
This testbed has been used to analyze the performance of our segmentation strategy and transport protocol (see following section).

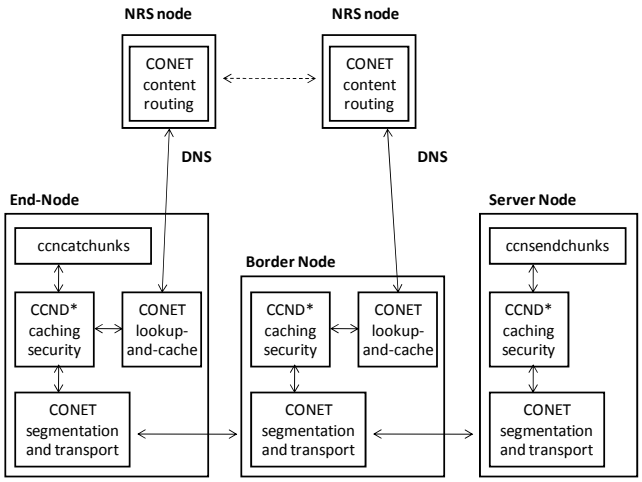

Figure 3. Implemented CONET nodes and modules

6. PERFORMANCE

In this section, we report the evaluation of the goodput achieved by ICTP and by the current CCNx implementation of CCN transport.
The CONET ICTP and the CCN congestion control are both receiver-driven. However, the data unit of CCN is the chunk, while CONET uses segments (i.e., carrier-packets). CCN does not have carrier-packets, and it uses IP fragmentation when a chunk is larger than the level 2 MTU (e.g. around 1500 bytes for ethernet). Moreover while ICTP mimics the TCP congestion control algorithms, the currently implemented mechanism in CCNx is a simple Go-Back-N ARQ rather than a TCP-like algorithm.
The experiment setup consists of two Linux PCs connected by a 100 Mbit/sec Ethernet (in this experiment we excluded the Border Node). On both PCs, the outgoing rate is limited to the rate of 10Mb/s at IP level by using the rate limiter of Linux traffic control (tc) module. A PC is the source of data (i.e., the ICN server) and the other PC is the sink of data (i.e., the ICN client that generates the interest messages). The size of the chunk header is 650 bytes for both CONET and CCN; the size of a segment is 1428 bytes.
In the first experiment, (see Figure 4), we measured the ICTP goodput in the transfer of a long file (10Mbyte). In order to show the performance gain in implementing the ICTP transport level segmentation, we compared the performance of the "multi segment" approach (the default in ICTP) with a solution in which each segment transports a chunk and IP fragmentation is used to fit the chunks into the Ethernet MTU. The comparison was run for different loss rates (reported in logarithmic scale in the x axis) and for different chunk sizes (4KB, 8KB, 16KB and 32 KB). For the sake of clarity, only the results for 4KB and 32 KB are plotted in Figure 4, as the results for 8KB and 16KB are intermediate between the results for 4KB and 32 KB. At least 10 runs for each case have been performed, and 95% confidence interval are plotted. Note that the lossless case (0% loss, i.e. the leftmost values in Figure 4) was arbitrarily plotted in correspondence of 10^{-4} loss, slightly abusing the logarithmic scale of x-axis.

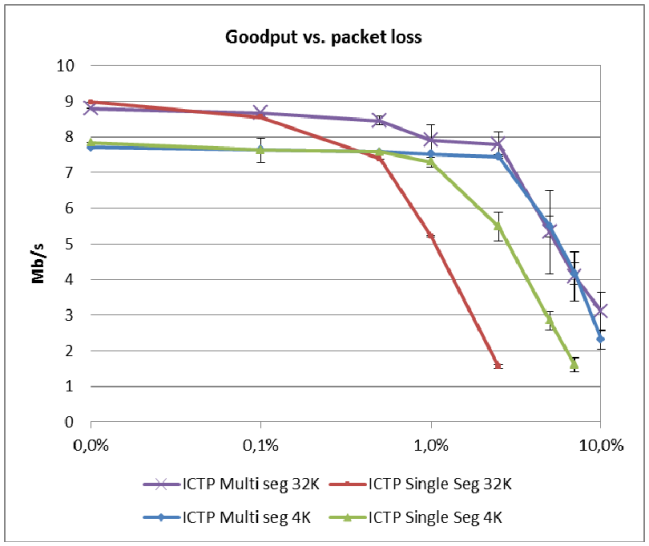

Figure 4. Multi segment vs. single segment goodput

In the lossless case the throughput of the single segment solution is slightly higher than the multisegment (for both 4KB and 32KB), because there is less overhead due to the ICTP carrier packets (we use a single carrier packet per chunk). The goodput for the 32 KB case is higher than the 4KB case because the chunk header overhead (650 bytes per chunk, mostly due to signature) has a smaller impact. When the loss rate increases, the performance of the single segment case decreases very sharply, because the loss of a single IP packet implies the loss of a whole chunk. In the single segment case, the loss seen at the congestion control level is higher than the loss at IP level, because the loss of a single IP fragment causes the loss of the whole chunk. This is more and more relevant when the chunk size is larger; thus, the goodput with 32 KB chunks becomes smaller than the goodput with 4 KB, as the loss rate increases. The impact of this effect is not limited to the throughput degradation that can be appreciated from Figure 4. As a matter of fact, also the fairness in the competition with TCP packets is impacted. In fact, for this reason an ICTP download using the single segment solution and competing with TCP flows will have a smaller goodput than TCP flows.
We have also compared the ICTP performances with the performances of the currently implemented congestion control in

CCNx (which may be a temporary solution). shows the application goodput measured for CCN and for ICTP versus the IP packet loss probability and for two different sizes of the data chunk (4K and 32K). In the ideal lossless case, the performance of the current CCNx transport implementation is slightly better, as the overhead of the carrier packet is saved. The performance of ICTP becomes better as soon as some packet loss is introduced. As noted above, this will also mean that if CCNx flows and ICTP flows compete on the same bottleneck, ICTP flows will be more aggressive and the share of resources would be unfair.

The possible coexistence of CCN or ICTP flows and TCP flows, which would happen in an evolutionary deployment of ICN, is further analysed in our technical report [14]. First, we analysed such coexistence by using a more abstract experiment, in which we did not use the actual ICTP protocol implementation. Rather, we used a plain source-driven TCP Reno in which we increased the data-units length to mimic the CCN chunk size (from 2K to 8K). We measured the goodput of a CCN flow that coexists with 10 regular TCP/IP. When we use small chunks, the IP goodputs are comparable: TCP/IP and CCN flows fairly share the link capacity. Conversely, when we use large chunks, CCN suffers the packet loss and TCP/IP flows tend to starve the CCN ones. This result indicates that the use of carrier-packets is necessary to allow ICN coexist with TCP/IP flows. We also investigated issues related to the coexistence of several CCN flows. We found that the large size of congestion control data units stretches the timescale of congestion control and produces significant oscillation of the goodput. This would be a problem, for instance, for non-real time streaming video services, which would require a large de-jitter buffer, penalizing the quality of experience.

We are now working on the analysis of the coexistence of real CCN flows, ICTP flows and TCP flows on the real testbed. Our preliminary results confirm that TCP flows and ICTP flows can starve CCN flows.

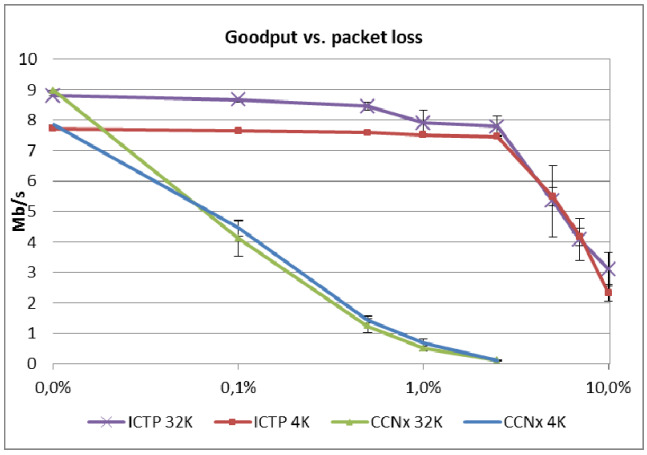

Figure 5. ICTP vs. existing CCNx transport

7. CONCLUSIONS

Data-centric security requires a valuable amount of per-chunk overhead and processing. To alleviate these drawbacks, we need large chunks. Conversely, large chunks are not effective to implement transport level flow control. Consequently, we propose to fragment chunks in carrier-packets and to use these new ICN data-units for forwarding and transport operations, while using chunks for security and caching operations.

We adapted the well-known TCP congestion control to the receiver-driven ICN context and we have implemented our solution in a testbed interoperable with CCNx based applications. The source code of the implementation is available under an open source license. The experimental performance results obtained in the testbed confirm our hypothesis on the need and feasibility of the proposed Information Centric Transport Protocol.

8. AKNOWLEDGEMENTS

This work was supported in part by the EU under the projects FP7–257123 “CONVERGENCE”.

9. REFERENCES

[1] D. Cheriton, M. Gritter, “TRIAD: a scalable deployable NAT-based internet architecture”, Technical Report (2000)

[2] T. Koponen, M. Chawla, B.G. Chun, A. Ermolinskiy, Kye Hyun Kim, S. Shenker, I. Stoica: “A data-oriented (and beyond) network architecture”, ACM SIGCOMM 2007

[3] V. Jacobson, et al., ”Networking named content”, in Proc. of ACM CoNEXT 2009

[4] D. Smetters, V. Jacobson: “Securing Network Content”, PARC technical report, October 2009

[5] D. Trossen, M. Sarela, and K. Sollins: “Arguments for an information-centric internetworking architecture” SIGCOMM Computer Comm. Review, vol. 40, 2010

[6] CONVERGENCE project website: www.ict-convergence.eu

[7] A. Detti, N. Blefari-Melazzi, S. Salsano, M. Pomposini, “CONET: A Content Centric Inter-Networking Architecture”, SIGCOMM Workshop ICN-2011, Toronto, Canada, August 2011

[8] A. Detti, S. Salsano, N. Blefari-Melazzi, “IPv4 and IPv6 Options to support Information Centric Networking”, draft-detti-conet-ip-option-03, Work in progress, May 2012

[9] A. Ghodsi, T. Koponen, B. Raghavan, S. Shenker, A. Singla, and J. Wilcox, "Information-Centric Networking: Seeing the Forest for the Trees", in Proc. of the 10th ACM Workshop on Hot Topics in Networks (HotNets-X), Cambridge, Massachusetts

[10] A. Kuzmanovic, E.W. Knightly “Receiver-Centric Congestion Control with a Misbehaving Receiver: Vulnerabilities and End-point Solutions”, Elsevier Comput. Network. 2007, 51, 2717–2737

[11] G. Carofiglio, M. Gallo, L. Muscariello, “ICP: Design and Evaluation of an Interest Control Protocol for Content-Centric Networking”, 1st Workshop on Emerging Design Choices in Name-Oriented Networking, co-located with IEEE Infocom 2012, Orlando, Florida USA

[12] CCNx project web site: www.ccnx.org

[13] S. Salsano, M. Cancellieri, A.Detti, “ICTP - Information Centric Transport Protocol for CONET ICN”, draft-salsano-ictp-00, Work in progress, May 2012

[14] N. Blefari Melazzi, M. Cancellieri, A. Detti, M. Pomposini, S. Salsano: “The CONET solution for Information Centric Networking”, Tech. report, available at http://netgroup.uniroma2.it/CONET.