Smart Transportation Systems (STSs) in Critical Conditions

Marco Cello, Cristiana Degano^, Mario Marchese*, Fabio Podda^*

**DITEN – University of Genoa, Genoa, Italy*
^Gruppo SIGLA S.r.l., Genoa, Italy
marco.cello@unige.it, cristiana.degano@grupposigla.it,
mario.marchese@unige.it, fabio.podda@grupposigla.it

I. Abstract

In the context of Smart Transportation Systems (STSs) in Smart Cities, the use of applications that can help in case of critical conditions is a key point. Examples of critical conditions may be natural disaster events such as earthquakes, hurricanes, floods, and manmade ones such as terrorist attacks and toxic waste spills. Disaster events are often combined with the destruction of the local telecommunication infrastructure, if any, and this implies real problems to the rescue operations.

The quick deployment of a telecommunication infrastructure is essential for emergency and safety operations as well as the rapid network reconfigurability, the availability of open source software, the efficient interoperability, and the scalability of the technological solutions. The topic is very hot and many research groups are focusing on these issues. Consequently, the deployment of a *Smart Network* is fundamental. It is needed to support both applications that can tolerate delays and applications requiring dedicated resources for real-time services such as traffic alert messages, and public safety messages. The guarantee of Quality of Service (QoS) for such applications is a key requirement.

In this chapter we will analyze the principal issues of the networking aspects and will propose a solution mainly based on Software Defined Networking (SDN). We will evaluate the benefit of such paradigm in the mentioned context focusing on the incremental deployment of such solution in the existing metropolitan networks and we will design a "QoS App" able to manage the Quality of Service on top of the SDN controller.

II. Introduction

In the last decades, we have seen a constant evolution of the ICT technologies that have brought us to an even more "connected" world and changed the way we use to interact with things and people. This evolution involves two main fields: computing and communication technology. Even if those two paradigms started independently, with the coming of the internet it is almost impossible to think about computation without networking and vice-versa. Nowadays the two paradigms are strictly related and this bond will probably strengthen in the future.

Figure 1 shows the evolution described above and highlights the beginning of the next ICT revolution, the Internet of Things (IoT) [Alcatel], where devices speak to each other and provide services, storage and computation in a heavily distributed environment.

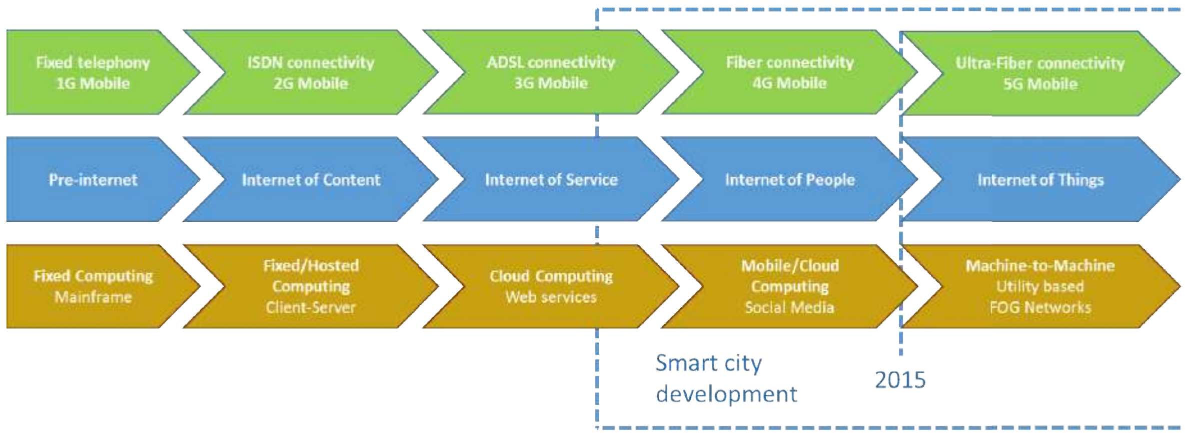

Fixed telephony 1G Mobile
ISDN connectivity 2G Mobile
ADSL connectivity 3G Mobile
Fiber connectivity 4G Mobile
Ultra-Fiber connectivity 5G Mobile
Pre-internet
Internet of Content
Internet of Service
Internet of People
Internet of Things
Fixed Computing Mainframe
Fixed/Hosted Computing Client-Server
Cloud Computing Web services
Mobile/Cloud Computing Social Media
Machine-to-Machine Utility based FOG Networks
Smart city development
2015

Figure 1. The evolution of Communication and Computing Technologies

As can be seen from Figure 1, along with the ICT evolution, new areas of research have emerged and new kinds of innovative services have brought us to the so-called "digital era". With the technology development, we have changed our way of life and most aspects of it are constantly influenced by this continuous progress. We are now in the era of "smart things", where technology helps us to perform new tasks or to optimize actions and behaviours we already carry on. A new area of interest that represents this kind of progress and involves, at large scale, citizens and their way of life is "Smart City".

The definition of a "Smart City" is not univocal and various shades of this term can be found. In [Chourabi14] there is a collection of different definitions of this term, but all of them agree with the fact that a Smart City is composed of a collection of interconnected computing technologies that cooperate to handle, in a smarter way, different aspects of the urban spaces (e.g. traffic and mobility, infrastructures, security and quality of life, etc.).

This field has emerged in 2005 in the United States [Ibm] and has progressively caught the attention of Universities, Enterprises and Governments. From the above description, and looking to Figure 1, it is clear why, currently, Smart Cities are considered as an emerging research topic: we have a very highly connected world, where computation is no more centralized, and where service-oriented architectures and Internet of Things are opening the door to new interesting challenges in distributed and collaborative computing.

There are many research initiatives and projects that attest the interest of the scientific community. In the last 3 years, the European Commission has allocated more than 350 Millions of Euro to the *Smart Cities and Communities* Work Programme and, concerning 2015, the call for projects on the same area in the Horizon 2020 framework has a budget of more than 100 Millions of Euro. Furthermore, we have to consider that there are many other projects that are financed directly from the local municipalities or from the institutions (research ministries and so on). In Italy, we can found a variety of initiatives, led by municipalities, devoted to collect and coordinate research projects in this field. Examples are: *Genova Smart City* [Genova], *Torino Smart City* [Torino] and *Milano Smart City* [Milano]. Moreover, it is estimated that in Italy the already-done investments in the Smart City research are of about 4 Billions of Euro.

Another interesting aspect of the presence of municipalities in these activities is that research results and prototypes can be easily deployed and tested directly into the cities, with positive effects for the visibility of the research institutions and for the chance to put a product on the market with less effort also reducing the time-to-market. This convergence of interest has brought to the constitution of “Living Labs”, open environments in the city for the design, test and validation of new products and services devoted, for example, to Intelligent Mobility. In a Living Lab, users can interact and experiment prototypes, providing very important feedbacks in terms of refinement and potential improvements.

As said before, the Smart City area covers different aspects of a city and involves different research topics, starting from citizens’ life to environment preservation through transportation and mobility. Each of them represents a sort of branch of the main topic and they are strictly related to each other. Actually every aspect of a city must take into account the other ones to maximise the final impact on the city itself.

The topic covered in this chapter concerns mobility, because traffic management and public transportation is one of the most important problem of a modern city and involves many of the above-mentioned aspects. When we talk about transportation systems, we need to consider how technologies have already improved them, and how new cutting-edge tools and paradigms can help to foster further progresses.

III. Network design for Smart Transportation Systems (STSs) in critical conditions

Intelligent Transportation Systems (ITS) can be defined as IT based applications which enable elements within the transportation system (i.e., traffic lights, roads, vehicles, etc.) to communicate with each other through wireless technologies, aiming to [Ezell10]:

- provide innovative services concerning different transport and traffic management modes;
- enable various users to be better informed;
- make safer, more coordinated, and 'smarter' use of transport networks;
- improve transportation system performance reducing congestion and increasing safety and traveller convenience.

The concept of transportation system is moving from Intelligent Transportation to Smart Transportation, also thanks to the advent of the Internet of Things (IoT) whose main characteristic is to link heterogeneous devices and technologies such as sensors, WiFi, RFID, smartphone, so giving the possibility to make transportations *cleverly smart* so to:

- increase driver and pedestrian safety through the development of applications such as real time traffic alerts, collision avoidance, cooperative intersection collision avoidance, crash notification systems and, thinking to the smart concept, Advanced Safety Vehicle (ASV) aimed at offering safer and smart driving via vehicle-to-vehicle communication;
- improve the operational performance of the transportation network, particularly by reducing congestion through applications acting on real time traffic data in order to, for example, optimize traffic signal lights and ramp metering;

- enhance personal mobility and convenience providing ICT-based applications able to provide the users (pedestrians and drivers) with real-time traveller information systems devoted to route selection and navigation capability;
- deliver environmental benefits that could be firstly come from the automotive point of view by equipping cars/motors with “Eco-driving” applications that optimize driving behaviour providing feedback to the driver on how to operate the vehicle [Ezel10].

As shown in Figure 2, Figure 3 and Figure 4, transportation and mobility have different impacts on a city, at different levels. A city that adopts an intelligent management of transportation and public mobility can benefit of different advantages, improve the quality of life, and preserve the environment.

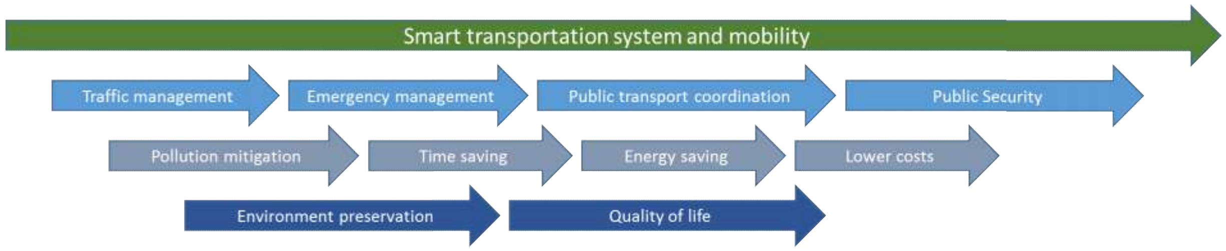


Figure 2. Involved topics in Smart Transportation Systems (STSs)

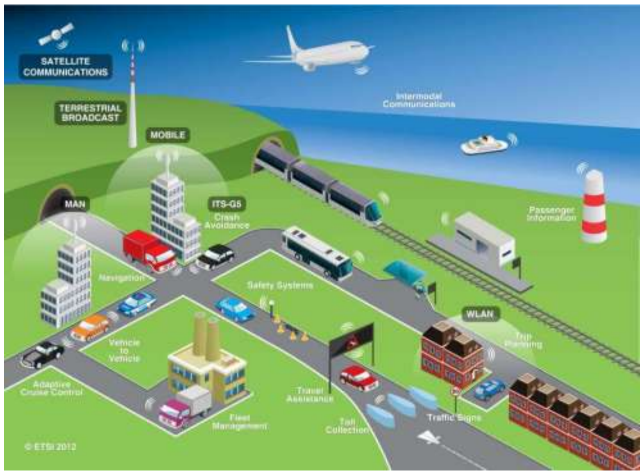


Figure 3. STS Scenario 1 (from [Etsi])

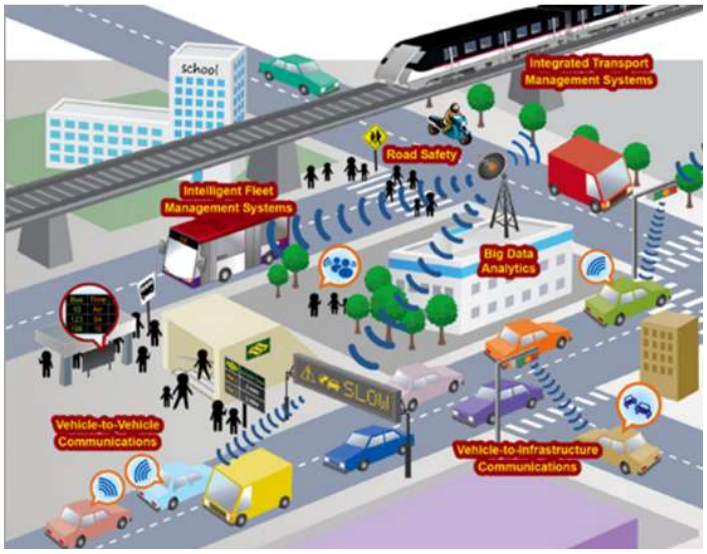

School
Integrated Transport Management Systems
Road Safety
Intelligent Fleet Management Systems
Big Data Analytics
SLOW
Vehicle-to-Vehicle Communications
Vehicle-to-Infrastructure Communications

Figure 4. STS Scenario 2 (from [Lta14])

According to the definition in [Lopez1], Smart Transportation Systems (STSs) use advanced and emerging technologies to deliver an end-to-end solution to transportation. A modern STS is composed of different actors shown in Table 1.

Sensors	**Sensors and networks of sensors. They can be fixed (e.g. traffic sensors, cameras, etc.) or mobile (e.g. vehicles);**
Decision Center	One or more **control and decision centres**, where all the data are collected and where algorithms are executed and decisions taken;
Actuators	A network of **actuators** (e.g. semaphores, mobile barriers, etc.) to remotely control traffic
Information Systems	Information systems to send messages to citizens/vehicles and public safety users (e.g. Variable Messages Panels, SMS services, etc.);
Users	Citizens; police or other institutions that interact with the STS to provide information or to request data; STS services that control all the sensors and actuators, gathering information or sending specific commands; other actors that use STS services;
Networks	Networks that interconnect all the elements mentioned above and guarantee a certain level of quality of service;

Table 1. Principal actors in a STS.

The next evolution of STSs takes into account the public transportation as an important part of mobility and is particularly devoted to the development of the new generation of the *Advanced Public Transportation Systems* (APTSs). APTSs are aimed at handling the main aspects of public and private mobility and at enhancing the concepts depicted in Figure 2.

Users and Main applications in an STS

Different kinds of users can access the network and receive different advantages from an STS:

Private users: for ordinary users, to be aware of traffic conditions could allow them to plan the trip, and to avoid congested routes and/or temporarily closed roads. Thanks to the availability of real time updates, private users will possibly change their route to avoid traffic jams created in the meantime, due, for example, to a road accident. In the following a possible list of user applications:

- intermodal route planner (APTS/Private/Mixed);
- availability / reservation / payment of parking areas;
- traffic information (accidents, congestion, etc.);
- information on road maintenance (e.g. street cleaning);
- other information such as taxi areas, bus stops and timetables, car sharing, mo-bike;
- proximity services such as notification entry in limited traffic zones, availability of parking spots, information centers, pharmacies or points of public interest, events, museums;

Institutional operators: STS are able to offer a significant added value to city authorities in the field of traffic management. Advantages range from the management of traffic flows to urban access management, from public transport to urban logistics. In more detail:

- supervision and control of traffic;
- crisis management;
- innovative services for City Logistics;
- information to local travelers, passengers and logistics operators;
- collection of information on user behaviors and new mobility patterns;
- analysis user satisfaction level with respect to mobility metrics;
- information service for users such as visitors, workers, landed vehicles, of given areas (not only the city but also the port);

Public transport operators: STS may improve:

- planning (frequency, timetables, type of vehicle);
- rescheduling for abnormal events;
- real time information provision to drivers;

Logistics operators: STSs include a wide range of applications not only for passengers but also for the freight sector. In this area principal STS applications include: electronic tolling, dynamic traffic management (management of variable speed limits, reservations and parking guidance, support for real-time navigation), real-time information systems, driver assistance such as warning systems, stability and driving style control. STS can also make easier the connection of different modes of transport, for example by means of integrated multimodal travel planning tools or of monitoring services for the co-modal transport of goods.

Public safety operators: the knowledge, through the STS, of the urban traffic situation, enables significant advantages to security operators like Civil Protection, Fire Department, and Police Forces who can choose the fastest route in terms of traffic to successfully complete their operations. The following table summarizes the different advantages depending on the operator category:

- civil protection: knowledge of the territory and definition of the risk scenario; definition of the defense structure identifying and mapping all the resources available to deal with the emergency; management of actions in case of emergency, through the Operations Room whose job is to coordinate the participation of municipal forces; provision of adequate information to citizens about the degree of risk exposure;
- firefighters: help provision during fires, uncontrolled releases of energy, sudden or threatening structural collapse, landslides, floods, and other public calamity, and, as in the previous case, definition of the defense structure identifying and mapping all the resources available to deal with the emergency; law enforcement Investigations of vehicles / people; support for intervention in the field;

STS Services control all sensors and actuators, gathering information or sending specific commands.

It is clear from the list above that the level of importance of the actors is not the same. We can reasonably argue that a citizen that requests an information to the STS service is less important than a police officer who is signalling some risk situations. Similarly, the operation of gathering information from sensors by the STS is typically more important than a routinary information request of an institution to the STS service. Starting from this point, we can define a priority class for each actor, which represents its "importance" inside the STS. Moreover, since the "importance" may depend not only on the actor but also on the type of request and on the particular situation (critical, normal, etc.), we can define an extended concept of priority considering all these aspects. In more detail, the priority can be seen within 3-dimensional space as depicted in Figure 5.

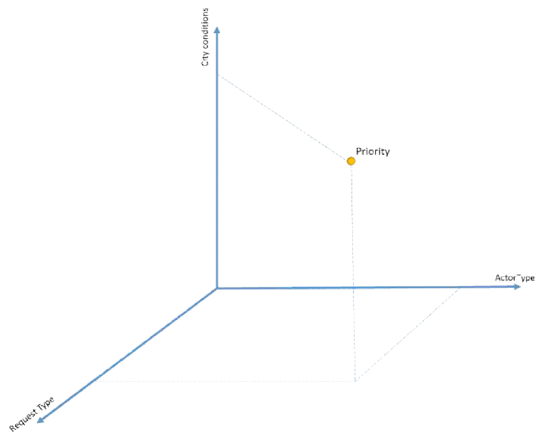


Figure 6. STS priority space.

Priority management helps provide different kind of service levels within different utilization scenarios. It is worth noting that such rules have a key role in emergency conditions, because if we do not have any problems in the infrastructure we can probably handle all the requests and services without any issue.

STS in critical conditions

As said above, the paradigm of Internet of Thins (IoT) is improving the level of intelligence and information of transportation systems allowing intelligent recognition, location, tracking and monitoring of mobility by exchanging information and communicating efficiently. This means that intelligent IoT–enabled transportation systems improve capacity, enhance travel experiences and make moving safer and more efficient; for example, emergency and other police services can use sensor networks along with smart traffic management to gain citywide visibility, to help alleviate congestion, and to rapidly respond to incidents [Lopez1].

On the other hand, existing IoT devices and related infrastructures have to face vulnerabilities and weaknesses that can affect STS efficiency. Paramount to the success of any transportation system is the efficiency in terms of safety and security, integrity, confidentiality and availability of information and services. Smarter systems, as highlighted above, can improve them. An STS can improve security by detecting and evaluating threats through the analysis of passenger information, electronic surveillance and biometric identification and ensuring in the same time that passenger and cargo data are only accessible to authorized personnel.

However, since all smart transportation solutions rely essentially on computing and networking, it is clear that the failure of one of these components represents a serious threat to the entire system. For this reason, resilience to failure is very important for a STS. It is a safety-critical system and wrong decisions can cause accidents and risks for the public security. In an STS, where almost all components are distributed, the failure of a single part is more than an exception. We can have different kind of failures depending on different causes; in this chapter, we consider failures depending on critical conditions.

We can define a critical condition as an exceptional situation where we have an emergency in a part of a city (or in the entire city) and where we probably have damages to infrastructures, things and people. The most popular critical conditions are caused by natural event such as floods, hurricanes and earthquakes, or by events such as terroristic attacks, blackouts, blazes, and environmental disasters. We have to take into account such events when implementing a STS in order to prevent situations where a damage of a part of our infrastructure can cause a general breakdown of the entire system. Furthermore, we have to consider that, in case of emergency or critical conditions, network loads may change unpredictably and quickly due to high amount of requests that users may perform, making the access to network services more difficult for users. So, we have to take into account these kinds of failures when we consider STSs:

Damages or failures of computational elements: regarding computational elements, we have to consider that a service interruption due to a failure can lead to different problems. For example, if the service that regulates road traffic goes down for some reason (e.g., hardware fault or blackout) we will probably have congestions and possible hazards for citizens. Countermeasures to these faults must be taken into consideration when we deploy the service. Concerning this aspect, we can reasonably say that today the trend is to host almost all important STS Services in a private datacenter or in a public cloud. Since modern datacenters massively use virtualization and data replication to ensure heavy fault-tolerance, one or more failures can be recovered without losing data or having service interruptions. Furthermore, datacenters are equipped with generators that guarantee electrical power even in case of blackouts and give the operators time to respond to these criticisms.

Damages or failures of sensors or actuators: damages or failures of sensors and actuators generally do not represent a serious threat to the entire STS and can be resolved in different way: -) to replicate sensors in critical position to ensure that a failure of a sensor can be recovered; -) to temporary replace sensor or actuators with human beings (e.g. the police) if the situation is critical.

Damages of network segments or failures of network devices: differently from sensors and actuators, a failure of the telecommunication network could represent a serious problem to the STS. Since we have seen that all the STS elements rely on the network (Figure 7), we can say that the communication infrastructure is a crucial point in the overall system, and its failure must be resolved as soon as possible avoiding connection interruptions and data loss. Damages to the network may occur in different places at different levels, due to the highly distributed nature of the network itself. Furthermore, damages at different levels cause different problems that a robust network must be able to tackle.

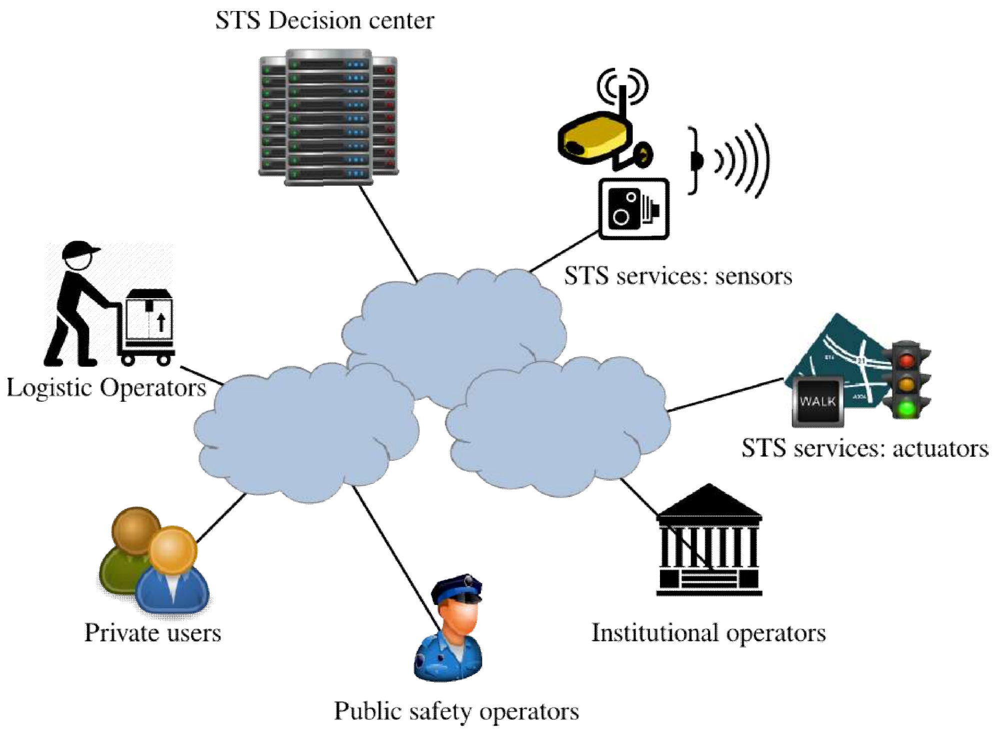


Figure 7. STS actors connected through telecommunication networks.

Examples of damage could be represented by a hurricane that gets down some antennas devoted to mobile communications, or a flood that damages the access network. In any case, we are facing with different problems and consequent different solutions. In some case reconfiguring the network to restore the service may be enough. Other situations may require to handle congestion due to the change of network topology and to the high network utilization. In other cases it may be important to guarantee to some “privileged” users the network access and ensure certain level of service.

The limits of current network technologies to support STS

Nowadays, networks and protocols are able to handle both failures and congestions, but the way they do does not completely respond to the requirements of traffic and services that run over an STS network. Moreover, the prioritization of the traffic and the Quality of Service (QoS) are not addressed, except for the Service Level Agreements (SLAs) that can be signed with Service Providers that are generally not available for wide-scale consumers. An example of the requirements of applications and services in a STS is shown in Table 3. These requirements have been defined within the PLUG-IN research project, funded by the Italian Ministry of Research and devoted to study and develop an infomobility system to support citizens and professionals. In the PLUG-IN project we have defined different scenarios of interaction between users and STS platform and we have studied different requirements of network and services related to the criticism level and the priority of the particular scenario. Scenarios are primarily organized depending on the type of user, required services and level of critical condition. In particular, as far as the level of critical condition is concerned, we distinguish different cases:

- low criticism, which refers to a situation of normal use of the STS, where there are no emergency events and urban traffic is below or at normal levels;
- middle criticism, which corresponds to a situation where the traffic has exceeded normal levels or a particular event happens for which it is necessary to act so to create minimum inconvenience to citizens or to prevent any damage to people and/or things;
- high criticism, which refers to a particular situation in which public safety is endangered and where you need to act quickly to resolve the emergency situation so to avoid or minimize damage to people and/or things. It is important to emphasize that the level of criticality is determined by the operation center on the basis of information from the field about the status of roads, of areas subject to risks such as rivers and landslides, and of industrial areas.

A description of the main QoS parameters is shown in Table 2:

- Availability, which represents the percentage of time for which the service is available compared to the total time;
- Time completion, also called "one-way delay," and defined as the time required to fulfill the user request;
- Information Loss, expressed as the rate of packet loss at the application level;

Scenario	Low/Middle criticism	High criticism
Private user that requires a path from A to B with an applicationon his/her smartphone	Availability: 99% Completion time: [5s – 15s] Information Loss: 0	Availability: ~90% Completion time: [25s – 40s] Information Loss: 0
Logistic operator user that has to fulfil some deliveries and requires the optimum path	Availability: 99,5% Completion time: [2s – 5s] Information Loss: 0	
STS that sends alert messages to the smartphones of citizens/professionals	Availability: ~99,9% Completion time: [10s]	Availability: ~99,999% Completion time: [5s]

to warn about a criticism in the city	Information Loss: 0	Information Loss: 0
Policeman that interacts with STS to resolve a criticism in the city	Availability: ~99,99% Completion time: [2s – 5s] Information Loss: 0	Availability: ~99,999% Completion time: [1s – 2s] Information Loss: 0
Institutional operators that want to send video/images to STS to document a particular situation **Institutional operators that want to view some video/images to follow a particular situation**	Availability: ~99,99% Video – Completion time: [2s] Video – Information Loss: ~10^{-2} Images – Completion time: [1s – 2s] Images – Information Loss: 0	Availability: ~99,999% Video – Completion time: [1s] Video – Information Loss: ~10^{-3} Images – Completion time: [1s – 2s] Images – Information Loss: 0

Table 3. Performance requirements of some STS users[1][2].

The requirements have been fixed depending on the emergency level of each scenario. Achieving most demanding requirements with the already deployed network is not so easy, especially the ones at high criticism level. Generally, we can say that, in critical conditions, networks can suffer and could not be able to handle services and connections to guarantee the given requirements, unless there is a particular Service Level Agreement (SLA) with the provider. Conversely, in normal conditions without any criticisms, we can argue that all above-mentioned scenarios will respect requirements.

In the traditional approach to networking, most of the functionality of the network is implemented within network nodes (switch/router). Within them the majority of the functionality is implemented in a dedicated hardware, such as an Application Specific Integrated Circuit (ASIC), that is an integrated circuit designed to solve a specific computation task. Unfortunately:

- ASICs that provide network capabilities have evolved slowly.
- The evolution of the functionality of ASICs is controlled by the manufacturer of the switch / router.
- The operating systems on board of network nodes are proprietary.
- Each node is configured individually.
- Operations such as provisioning, change management and de-provisioning are time consuming and subject to errors.

In addition, the following features mean that the world of the implemented networks is somehow "crystallized":

- Currently used routing protocols (RIP, OSPF, BGP) were developed for the most part 20 years ago and did not evolve except for few differences.
- Mechanisms for Traffic Engineering are difficult to implement and the quality of service is difficult to achieve.

[1] ITU-T G.1010, End-user multimedia QoS categories, 11/2001.
[2] Optionally, it can be included also an indication of the user/service priority. This priority is intended as a parameter to use in case of Call Admission Control (CAC). For example, in case of network outage due to catastrophic events, STS could serve the requests from public safety users before than the one of private users due to the high priority characterization.

Current networks are difficult to manage and, although distributed control allows a certain scalability, they can be a source of problems when it is necessary to reconfigure them in order to achieve specific quality objectives (eg. Traffic Engineering for specific traffic flows). When a network device reveals a network topology change (i.e. for a fault on a line or on a device), it can change its configuration to re-route incoming traffic through another path in the network by using distributed routing protocols such as BGP and OSPF. Although this mechanism is quite efficient (it takes few seconds to re-configure a network), it can be improved. For example it does not take into account overall network topology and link utilization. It may happen that, in case of high traffic and network failure, the reconfiguration addresses the traffic to an already congested network portion. In this case, the congestion will worsen and many packets will be dropped.

Another problem with currently used networks is that generally all the traffic is routed following the *best-effort* strategy. In this situation the traffic is handled without any type of priority and if there is network congestion no priority level will privilege important traffic. Signing an SLA with service provider is the only way to handle this kind of problems, but, in an STS environment, where we have lot of different actors and different Internet Service Providers (ISPs), this strategy is not applicable. Moreover, we have to take into account that SLAs are not thought to be dynamic and service levels can hardly be varied over time. A further problem is that with the current networks is practically impossible to quickly deploy new applications or protocols within the router since, as mentioned earlier, they implement proprietary operating systems.

Towards a new network architecture

Along with the evolution of ICT and services that massively use it, networks such as the Internet have increasingly become critical infrastructures, because their failure compromise the entire systems that rely on them. Furthermore, with the IoT and its future evolution such as the Internet of Everything (IoE), the Internet will become even more important and will represent a critical aspect of our life.

Keeping this in mind, we can consider Smart Transportation Systems as critical systems that use a critical infrastructure, the network, which has to be designed to be fault-tolerant to critical conditions. Obviously, this network has to be as secure as possible, i.e. the network must be able to contrast cyber attacks, such as Distributed Denial of Service (DDoS), and traffic injection. We can summarize network fault tolerance and security requirements with the term survivability, which is the capability of a system to fulfil its mission, in a timely manner, in the presence of threats such as attacks or natural disasters [Sterbenz10].

The need to ensure QoS performance and adequately support the applications and services of a STS, pushes the use of network architectures that are different from those in place. This is true in particular regarding the dichotomy **data-plane/control plane**. In this sense, the main characteristics which a new generation architecture must meet are:

- Improving the management of the network: faster handling, targeted interventions to network elements and global network view in terms of topology, and real time traffic.
- Implementing end-to-end Traffic Engineering policies.
- Assuring the dynamic allocation of network resources for Network Function Virtualization (NFV).
- Facilitating the evolution of faster network functionalities, based on the life cycle of software development.

In the next sections, we will go deep in these problems trying to explore the innovative solutions that can respond to these needs, highlighting the problems that exist on the already-deployed networks and how to cope with them.

IV. Software-Defined Networking

Current computer networks are complex and difficult to manage. They use with many kinds of devices, from routers and switches to middleboxes such as firewalls, network address translators, server load balancers, and intrusion detection systems. Routers and switches run distributed control software that is typically closed and proprietary. On the other hand, network administrators typically configure individual network devices by using configuration interfaces that vary across vendors and even across different products from the same vendor. This operation mode has slowed innovation, increased complexity, and inflated operational costs to run a network.

Software Defined Networking (SDN) is a new paradigm in networking that is revolutionizing the networking industry by enabling programmability, easier management and faster innovation. These benefits are made possible by its centralized control plane architecture, which allows the network to be programmed by the application and controlled from one central entity.

The defining feature of SDN is its large scale adoption in industry [Jain13, Nicira12], especially as compared with its predecessors. This success can be attributed to a perfect storm of conditions among equipment vendors, chipset designers, network operators, and networking researchers [Feamster14]. Here the principal reasons of the transformation from designed networks to programmable networks [Shenker12]:

- Networks are hard to manage: whereas computation and storage resources have been virtualized by creating a more flexible and manageable infrastructure, networks are still notoriously hard to manage. In facts network administrators need large share of sys-admin staff.
- Networks are hard to evolve especially compared with the ongoing and rapid innovation in system software (e.g. new languages, operating systems, ...). On the other hand, networks are stuck in the past. For example routing algorithms change very slowly, as highlighted before, and network management is extremely primitive.
- Network design is not based on formal principles. Whereas operating systems courses teach fundamental principles like mutual exclusion and other synchronization primitives (e.g. files, file systems, threads, and other building blocks), networking courses teach a big bag of protocols with no formal principles, and just general design guidelines.

Brief History

The problems described before exist from the dawn of networking. Making computer networks more programmable makes innovation in network management possible and lowers the barrier to deploy new services. This section is dedicated to the review of early efforts on programmable networks. This section follows in large part the work published in [Feamster14].

SDN has gained significant traction in the industry. Although the excitement about SDN has become more palpable fairly recently, many of the ideas underlying the technology have evolved over the past 20 years. SDN resembles past research on active networking, which articulated a vision for programmable networks, albeit with an emphasis on programmable data planes. SDN also relates to previous work on separating the control and data planes in computer networks.

Active Networking

Active networking represented a new radical approach to network control by envisioning a programming interface through network API that exposed resources (e.g., processing, storage, and packet queues) on individual network nodes and supported the construction of functionalities to apply to the packets inside the router. Active networking community pursued two programming models:

- the capsule model, where the code to execute at the nodes was carried in-band in data packets [Wetherall98];
- the programmable router/switch model, where the code to execute at the nodes was established by out-of-band mechanisms. [Bhattacharjee97, Smith96].

Active networks offered intellectual contributions that relate to SDN. In particular the research in active networks pioneered the notion of programmable networks as a way of lowering the barrier to network innovation. Moreover the need to support experimentation with multiple programming models led to work on network virtualization. Finally, early design documents cited the need to unify the wide range of middlebox functions with a common, safe programming framework.

Separating Control and Data Planes

As the Internet flourished in the 1990s, the link speeds in backbone networks grew rapidly, leading equipment vendors to implement packet-forwarding logic directly in hardware, separate from the control-plane software. Moreover, the rapid advances in commodity computing platforms meant that servers often had substantially more memory and processing resources than the control-plane processor of a router deployed just one or two years earlier. These trends catalyzed two innovations:

- an open interface between control and data planes, such as the ForCES (Forwarding and Control Element Separation) [Yang04] interface standardized by the IETF and the Netlink interface to the kernel-level packet-forwarding functionality in Linux [Salim03];
- a logically centralized control of the network, as seen in the RCP (Routing Control Platform) [Caesar05, Feamster04] and SoftRouter [Lakshman04] architectures, as well as the PCE (Path Computation Element) [Farrel06] IETF protocol.

Moving control functionalities off the network equipment and into separate servers (selecting better network paths, minimizing transient disruptions) represented a paradigm-shift from the Internet's design making sense because network management is, by definition, a network-wide activity. Logically centralized routing controllers [Caesar05, Lakshman04, vanderMerwe06] were made possible by the emergence of open-source routing software [Bird, Handley05, Quagga] that lowered the barrier to create prototype implementations. To broaden the vision of control- and data-plane separation, researchers started exploring clean-slate architectures for logically centralized control: 4D project [Greenberg05] and Ethane project [Casado07].

Why Didn't it Work Out?

Although active networks articulated a vision of programmable networks, the technologies did not see widespread deployment due to the paradigm shift of active networks research compared to the Internet community. Moreover the lack of an immediately compelling problem or a clear path to deployment was a block for the large adoption of the mechanisms proposed by active networks research.

Concerning the separation of control plane from data planes, the dominant equipment vendors had little incentives to adopt standard data-plane APIs such as ForCES, since open APIs could attract new entrants into the marketplace. At the end, although industry prototypes and standardization efforts made some progress, widespread adoption remained elusive.

The ideas underlying SDN faced a tension between the vision of fully programmable networks and pragmatism that would enable real-world deployment. OpenFlow, described later, is a balance between these two goals by enabling more functions than earlier route controllers and building on existing switch hardware. Although relying on existing switch hardware did somewhat limit flexibility, OpenFlow was almost immediately deployable, allowing the SDN movement to be both pragmatic and bold. The creation of the OpenFlow API [McKeown08] was followed quickly by the design of controller platforms such as NOX [Gude08] that enabled the creation of many new control applications.

Theoretical Aspects

In networking we can envision two "planes" [Wu82]:

- **Data plane**: devoted to process and deliver packets by using the information of local forwarding state (e.g. routing entries);

  Forwarding state + packet header → Forwarding decision

- **Control plane**: devoted to compute the status in routers. For example, it determines how and where packets are forwarded, takes decisions about traffic engineering, firewalls, …

In current network devices (e.g., routers and switches), see Figure 7, the control plane is implemented in distributed protocols or directly through a manual configuration of network devices.

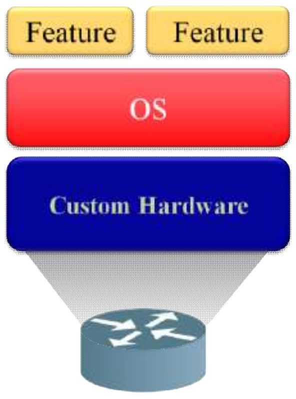


Figure 8. Stack of current network devices.

The two different planes (data and control) would require different abstractions. As far as data plane is concerned, the abstraction is well-known and is the protocol stack (ISO/OSI or TCP/IP). Applications are built on reliable (or unreliable) transport (e.g. TCP, UDP) that, in turn, are built on a best-effort global packet delivery protocol (e.g. IP). IP is then built on best-effort local frame delivery that uses a local physical transfer of bits. TCP/IP protocol stack is shown in Figure 8.

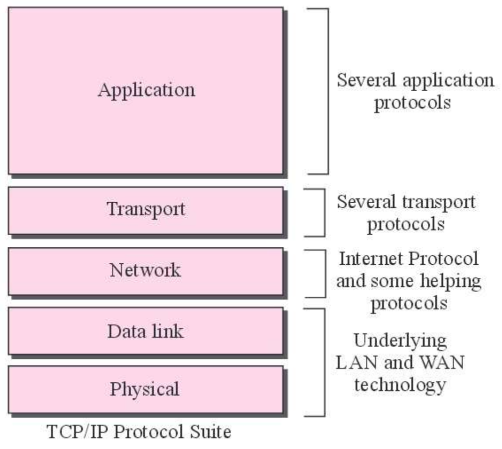

Application
Several application protocols
Transport
Several transport protocols
Network
Internet Protocol and some helping protocols
Data link
Physical
Underlying LAN and WAN technology
TCP/IP Protocol Suite

Figure 9. TCP/IP protocols stack/suite.

Many mechanisms belong to the control plane functionalities. Some of them are access control lists (ACLs), Virtual LAN, firewall, traffic engineering mechanisms (adjusting weights, MPLS). Unfortunately, there is no abstraction for the control plane.

Control plane must compute forwarding state. To accomplish its task, the control plane must:

- figure out what network looks like → **topology discover**;
- figure out how to accomplish the goal on a given topology → **accomplish the goal**;
- tell the switches what to do → **configure forwarding state**.

Currently, when we want design and implement a new control plane functionality we view the three tasks as a natural set of requirements and we require each new protocol to solve all three. Obviously **two of these tasks can be "reused"** for any new control plane functionality we want to implement. In particular:

- Determining the topology information
- Configuring the forwarding state on routers/switches

In other words, if we implemented the mechanisms able to determine the topology and configure the forwarding state in network devices we could reuse the same mechanisms for any new control functionality. This is the theoretical core idea of SDN:

"SDN is the use of those two control planes abstractions."

Abstraction 1 – **Global Network View**:

- the global network view provides information about the current network and is then devoted to the topology discover;
- its implementation is “Network Operating System” that runs on a centralized server in network (replicated for reliability);

Abstraction 2 – **Forwarding Model**:

- the forwarding model provides a standard way to define the forwarding state inside network devices;
- a common implementation is OpenFlow (described later).

In current networks, traditional control mechanisms (e.g. routing, traffic engineering, multicast) run in a distributed fashion as shown in Figure 9.

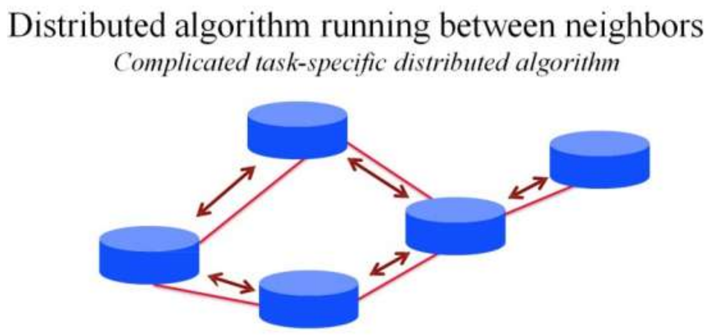

Distributed algorithm running between neighbors
Complicated task-specific distributed algorithm

Figure 10. Traditional Control Mechanisms

In SDN, as shown in Figure 11, Network OS that rusn on a server, communicates with each SDN device through a standard mechanism (Forwarding model), builds the Global Network View (a graph) and makes visible the resources of the network (in terms of a graph) to all control programs that run on Network OS.

With SDN, the control plane functionalities are not distributed control mechanisms but simple pieces of software that run on graphs (built by network OS).

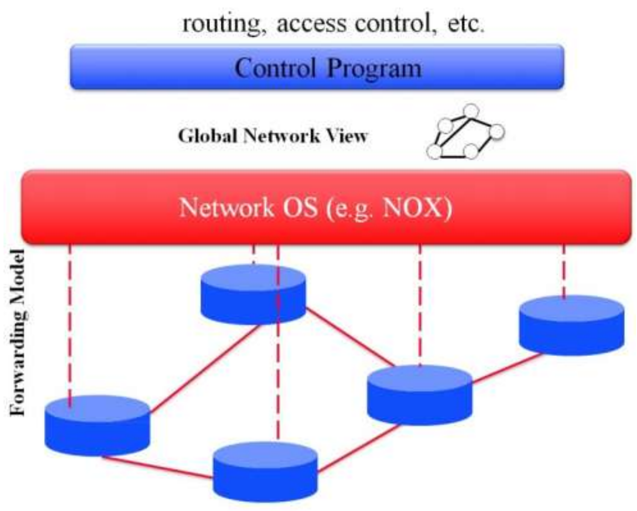


Figure 11. SDN "Layers" for Control Plane.

A clean separation of concepts is reached through SDN.

- Control Programs (e.g. router mechanisms, TE schemes): to express their goals on a Global Network View (not a distributed protocol but a graph algorithm);
- Network OS: to build the Global Network View through the communication with switches/routers. It also conveys configurations from the control program to switches;
- Router/switches: which merely follows the orders from NOS;

There is a clean separation of control and data planes that will not be packaged together in proprietary boxes. SDN enables the use of commodity hardware (network devices) and 3rd party software (network OS and control programs). Obviously, abstractions don't eliminate complexity, but now every system component is tractable. Network OS are still complicated pieces of code but Network OS is reusable for every control program.

Other aspects and details can be found in [Shenker11, Shenker12, Shenker13, McKeown11].

SDN in practice

As said before, Software Defined Networking (SDN) is a new networking paradigm that is revolutionizing the networking industry by enabling programmability, easier management and faster innovation. These benefits are made possible by its centralized control plane architecture (network OS), which allows the network to be programmed by the application and controlled from one central entity.

SDN architecture is composed of both switches/routers and of a central controller (SDN controller or network OS), as in Figure 11. The SDN device processes and delivers packets according to rules stored in its flow table (forwarding state), whereas the SDN controller configures the forwarding state of each switch by using a standard way. The controller is also responsible to build the virtual topology representing the physical one. Virtual topology is used by application modules that run on top of the SDN controller to implement different control logics and network functions (e.g. routing, traffic engineering, firewall state).

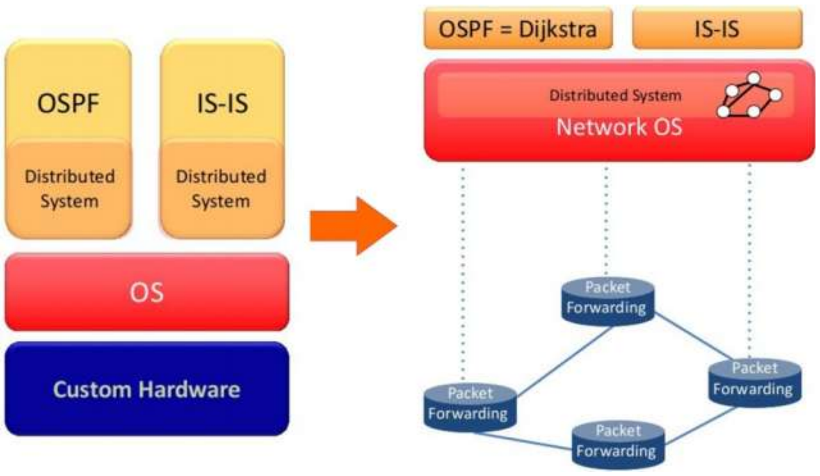


Figure 12. Legacy router and SDN architecture.

Implementation Aspects: Forwarding Model

An implementation of the forwarding model and a standard de facto in SDN is Openflow (OF) [Openflow15]. In OF the forwarding model is implemented as a <match-actions> as shown in Figure 12.

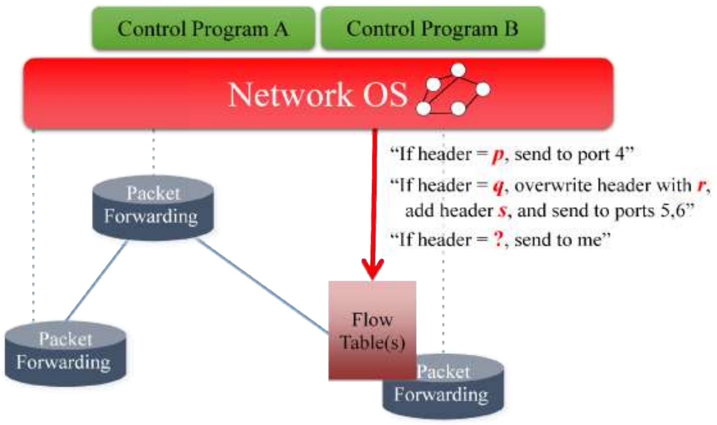

Control Program A
Control Program B
Network OS
"If header = p, send to port 4"
"If header = q, overwrite header with r, add header s, and send to ports 5,6"
"If header = ?, send to me"
Packet Forwarding
Packet Forwarding
Flow Table(s)
Packet Forwarding

Figure 13. Basic of forwarding model in OF.

In more detail, the OpenFlow architecture is illustrated in Figure 13 [Nunes14]. The forwarding device, or OpenFlow switch, contains one or more flow tables and an abstraction layer that securely communicates with a controller via the OpenFlow protocol. Flow tables consist of flow entries, each of which determines how packets belonging to a flow will be processed and forwarded. Flow entries typically consist of:

1. Match fields, or matching rules, used to match incoming packets; match fields may contain information found in the packet header, ingress port, and metadata;
2. counters, used to collect statistics for the particular flow, such as the number of received packets, the number of bytes and the duration of the flow;
3. a set of instructions, or actions, to be applied upon a match; they dictate how to handle matching packets.

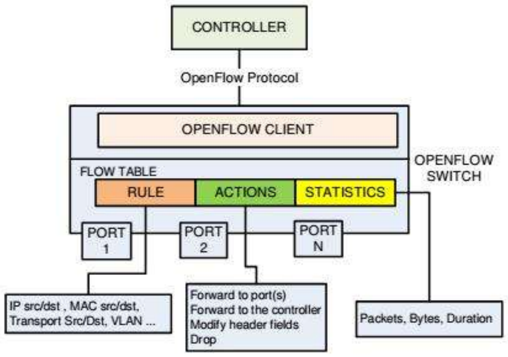
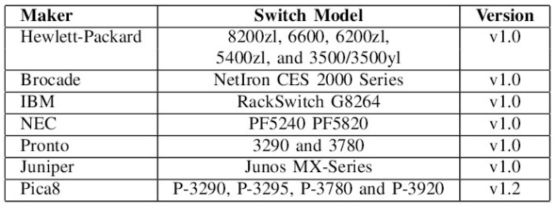


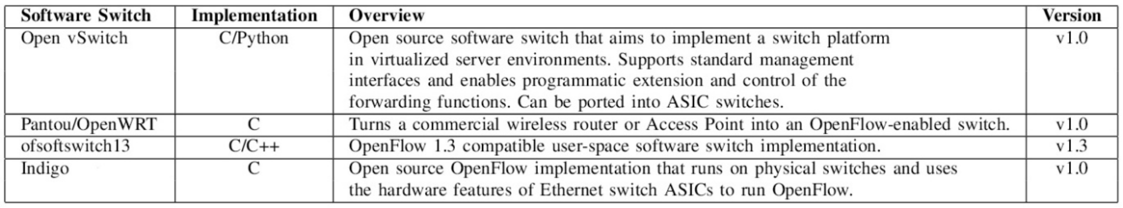
Figure 14. OF device.

Upon a packet arrival at an OpenFlow switch, packet header fields are extracted and matched against the matching fields portion of the flow table entries. If a matching entry is found, the switch applies the appropriate set of instructions, or actions, associated with the matched flow entry. If the flow table look-up procedure does not result on a match, the default action is to send the packet to the SDN controller that will take the decision and will install the rules on OF devices. The communication between controller and switch happens via OpenFlow protocol, which defines a set of messages that can be exchanged between these entities over a secure channel. Using the OpenFlow protocol a remote controller can, for example, add, update, or delete flow entries from the switch's flow tables. That can happen reactively (in response to a packet arrival) or proactively. Figure 14 and 15 contain, respectively, main current available commodity switches by makers and current software switch implementations, both compliant with the Openflow standard.



Maker	Switch Model	Version
Hewlett-Packard	8200zl, 6600, 6200zl, 5400zl, and 3500/3500yl	v1.0
Brocade	NetIron CES 2000 Series	v1.0
IBM	RackSwitch G8264	v1.0
NEC	PF5240 PF5820	v1.0
Pronto	3290 and 3780	v1.0
Juniper	Junos MX-Series	v1.0
Pica8	P-3290, P-3295, P-3780 and P-3920	v1.2

Figure 15. Main current available commodity switches by makers, compliant with the Openflow standard.



Software Switch	Implementation	Overview	Version
Open vSwitch	C/Python	Open source software switch that aims to implement a switch platform in virtualized server environments. Supports standard management interfaces and enables programmatic extension and control of the forwarding functions. Can be ported into ASIC switches.	v1.0
Pantou/OpenWRT	C	Turns a commercial wireless router or Access Point into an OpenFlow-enabled switch.	v1.0
ofsoftswitch13	C/C++	OpenFlow 1.3 compatible user-space software switch implementation.	v1.3
Indigo	C	Open source OpenFlow implementation that runs on physical switches and uses the hardware features of Ethernet switch ASICs to run OpenFlow.	v1.0

Figure 16. Current software switch implementations compliant with the Openflow standard.

Implementation Aspects: Controller

The SDN controller has been compared to an operating system in [Gude08], in which the controller provides a programmatic interface to the network that can be used to implement management tasks and offer new functionalities. As a practical example of the layering abstraction accessible through open application programming interfaces (APIs), Figure 16 illustrates the architecture of an SDN controller based on the OpenFlow protocol. This specific controller is OpenDaylight controller [Opendaylight].

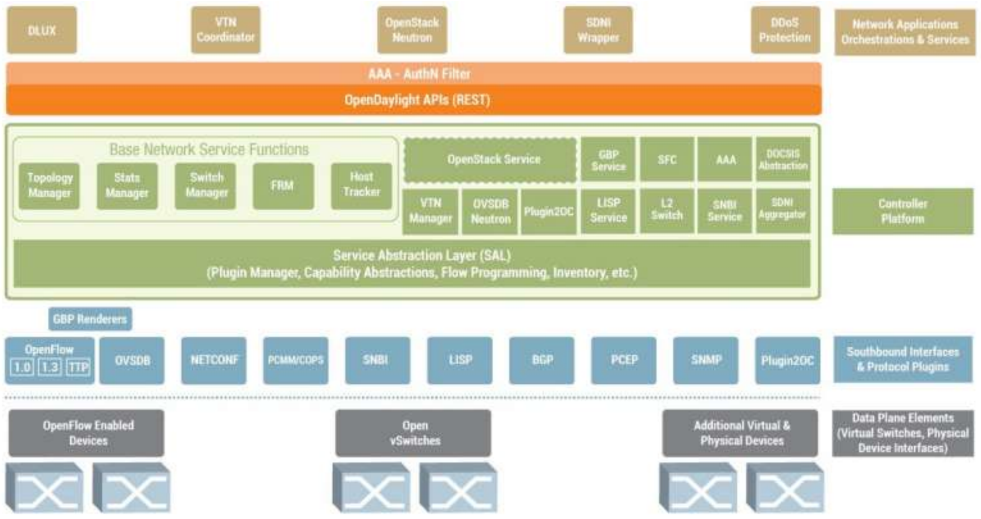
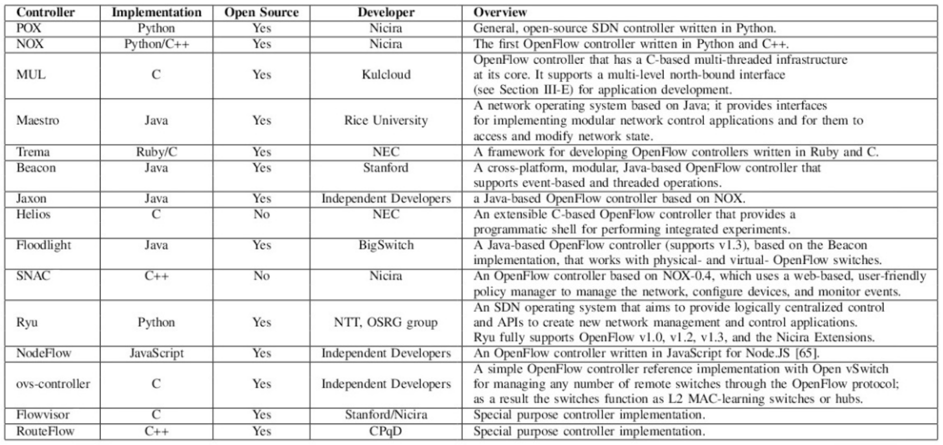


Figure 17. OpenDaylight controller architecture.

It is possible to observe the separation between the controller and the application layers. Applications can be written in Java and can interact with the built-in controller modules via a OpenDaylight API. Other applications can be written in different languages and interact with the controller modules via the REST API. Figure 17 contains current controller implementations compliant with the Openflow standard.



Controller	Implementation	Open Source	Developer	Overview
POX	Python	Yes	Nicira	General, open-source SDN controller written in Python.
NOX	Python/C++	Yes	Nicira	The first OpenFlow controller written in Python and C++.
MUL	C	Yes	Kulcloud	OpenFlow controller that has a C-based multi-threaded infrastructure at its core. It supports a multi-level north-bound interface (see Section III-E) for application development.
Maestro	Java	Yes	Rice University	A network operating system based on Java; it provides interfaces for implementing modular network control applications and for them to access and modify network state.
Trema	Ruby/C	Yes	NEC	A framework for developing OpenFlow controllers written in Ruby and C.
Beacon	Java	Yes	Stanford	A cross-platform, modular, Java-based OpenFlow controller that supports event-based and threaded operations.
Jaxon	Java	Yes	Independent Developers	a Java-based OpenFlow controller based on NOX.
Helios	C	No	NEC	An extensible C-based OpenFlow controller that provides a programmatic shell for performing integrated experiments.
Floodlight	Java	Yes	BigSwitch	A Java-based OpenFlow controller (supports v1.3), based on the Beacon implementation, that works with physical- and virtual- OpenFlow switches.
SNAC	C++	No	Nicira	An OpenFlow controller based on NOX-0.4, which uses a web-based, user-friendly policy manager to manage the network, configure devices, and monitor events.
Ryu	Python	Yes	NTT, OSRG group	An SDN operating system that aims to provide logically centralized control and APIs to create new network management and control applications. Ryu fully supports OpenFlow v1.0, v1.2, v1.3, and the Nicira Extensions.
NodeFlow	JavaScript	Yes	Independent Developers	An OpenFlow controller written in JavaScript for Node.JS [65].
ovs-controller	C	Yes	Independent Developers	A simple OpenFlow controller reference implementation with Open vSwitch for managing any number of remote switches through the OpenFlow protocol; as a result the switches function as L2 MAC-learning switches or hubs.
Flowvisor	C	Yes	Stanford/Nicira	Special purpose controller implementation.
RouteFlow	C++	Yes	CPqD	Special purpose controller implementation.

Figure 18. Current controller implementations compliant with the Openflow standard.

V. QoS Applications for STS

As said before, many applications nowadays rely on Quality of Service (QoS) guarantees. Some of them are telemedicine, tele-control, tele-learning, telephony, video-conferences, online gaming, multimedia streaming and applications for emergencies and security. Each application, having very different characteristics, needs a specific degree of service, defined at the application layer. As described previously, applications and services in STS mainly rely on the satisfaction of QoS requirements.

In order to guarantee specific QoS requirements, QoS management is strongly necessary. QoS management functions are aimed at offering the necessary tools to pursue this objective. A possible classification of the QoS Management functions is shown in Figure 18, from [Marchese07]. Others classifications can be found in [Aurrecoechea98, Campbell94, Hong03].

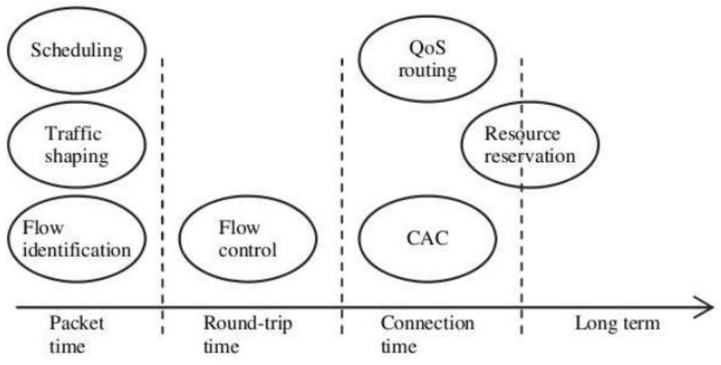


Figure 19. QoS management functions versus time.

Traditionally QoS management has been left to the end-hosts by virtue of "end-to-end principle" [Saltzer81] by using end-to-end congestion control (TCP) [Afanasyev10]. In-network solutions in legacy networks are proposed in MPLS-TE [Awduche01, Applegate03], in OFPS-TE [Katz03] and Segment Routing [Filsfils2015] but these approaches rely on distributed control architectures that may limit the control power of the entire network.

As described before, the ability of the SDN controller to receive (soft) real-time information from SDN devices and to make decisions based on a global view of the network, coupled with the ability of ''custom''-grained flow aggregation inside SDN devices, makes Traffic Engineering (TE) one of the most interesting use cases for SDN networks.

STS scenario

In a STS scenario (Figure 20) different actors (described in Section III) use the network (here simplified and composed by 8 SDN devices).

In case of **middle criticism** conditions, we need to give higher priority to certain actors. As clear from the scenario in Figure 20, public safety operators, institutional operators, actuators, sensors and STS decision center generate **high priority flows** (in red), logistic operators generate **mid priority flows** (in yellow) and private users generate **low priority flows** (in green).

Each SDN device is configured with multi-queues: in each OF switch one or more queues can be configured for each outgoing interface and used to map flow entries on them. Flow entries mapped to a specific queue will be treated according to the queue's configuration in terms of service rate.

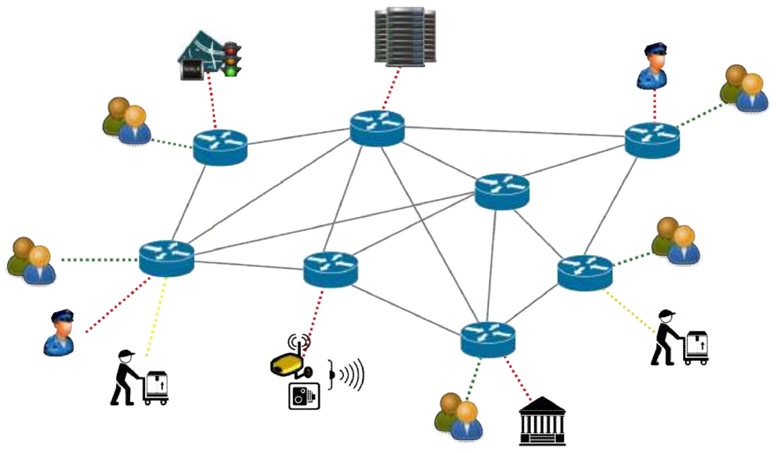


Figure 20. STS Scenario acting through SDN.

In particular, let us suppose that for each interface of each OF device, there are two assigned queues, each of them dedicated to a specific type of traffic with a predefined service rate. The first queue (q_0) of each OF device is assigned to **high priority flows** while the second one (q_1) is dedicated to **low priority flows**. **Mid priority flows** are assigned to the first queue at first instance. See Figure 20. If a mid priority flow traversing q_0 suddenly increases its rate (i.e., they are no longer conformant with QoS constraints on which the OF device has been configured), because, for example, the user wants an added value service, the queue will start to grow and it could end up losing packets. See Figure 21. The performance of the other flows (high priority flows) traversing the queue will be affected by this event, both in terms of delay and packet loss.

A possible solution that copes with a limitation of OF is traffic re-routing (TR). By TR we can change the path/route of a flow (or a subset of them) in order to reduce imminent congestion, to avoid link disruption and to improve the QoS. A key factor of TR is time: time that elapses between the congestion event and the reroute of the flow should be as small as possible. This problem has been tackled in [Boero15a, Boero15b].

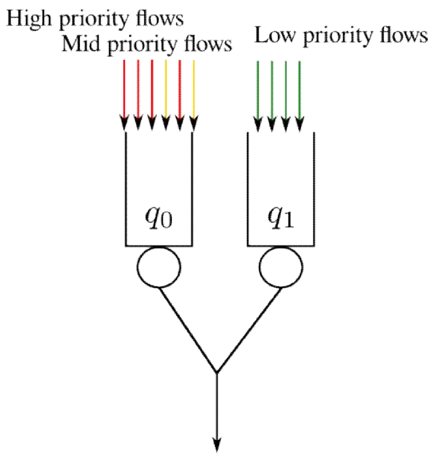


Figure 21. Queue model of OF device.

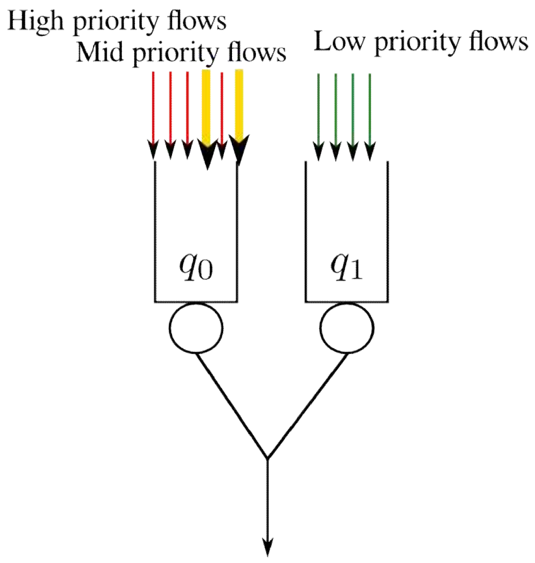


Figure 22. Some of mid priority flows increases their rate are no longer conformant.

We propose a strategy that limits this effect, ensuring that flows that prove to be compliant with the QoS constraints can exploit the needed bandwidth without suffering from any performance degradetion. This solution has the task to identify the traffic which is not conformant to the rate constraints and re-route it (or drop it, if needed), in order to avoid the degradation of the quality experienced by other flows traversing the network. Since we want to devise a solution which is compatible with any underlying hardware, we design and implement the strategy inside the SDN controller. We chose Beacon [Erickson13] as SDN controller. Beacon is a multi-threaded Java-based controller that relies on OSGi and Spring frameworks and it is highly integrated into the Eclipse IDE. In spite of a specific choice of the controller, our modifications can be implemented in any controller. The principal modifications of Beacon are:

Statistics Polling Beacon periodically sends statistic requests to the switches. The statistics requested are flow, port and queue statistics. In addition to the basic statistics that the OpenFlow protocol 1.0 makes available, we added specific functions to the controller, which allow Beacon to exploit the collected data in order to compute the parameters useful to apply the chosen strategy. The statistics computed by the controller are shown in Table 3. The main extracted feature is the Estimated Rate (ER) for ports, queues and flows.

Table 4. BeaQoS statistics.

Statistics available in OpenFlow 1.0		Statistics computed by BeaQoS
Tx Bytes per Flow	$\rightarrow$	Estimated Rate per Flow
Tx Bytes per Port	$\rightarrow$	Estimated Rate per Port
Tx Bytes per Queue	$\rightarrow$	Estimated Rate per Queue
Flow Match Flow Actions Queue ID	$\rightarrow$	Flows per Queue

Routing This module has been modified to the purpose of implementing the proposed algorithms. When a switch receives a new flow, it contacts the controller in order to know where to forward the traffic. When the controller has to assign each flow to a specific queue, it checks a variable that identifies the algorithm to run. Beacon performs a routine to select the correct queue based on the chosen strategy and then notifies the node through the installation of a flow modification.

The scenario in which we present our solution involves a class-based system in which flows are identified by traffic descriptors. The issue we want to investigate deals with a flow characterized by a specific rate limit, which, for some reason, violates this constraint. At this point our system recognizes the problem and re-routes the flow in a more suitable queue, in order to avoid traffic congestion and quality degradation. We suppose two main types of traffic:

- **High priority (HP) and Mid priority (MP)** - characterized by a rate not exceeding 100 kbit/s;
- **Mid priority (MP) not conformant** - characterized by a rate that sometimes can exceed 100 kbit/s;
- **Low priority (LP)** - displaying a rate greater than 100 kbit/s, most of the time.

We introduce and implement a solution that will be called **Conformant**, to the purpose of re-establishing the correct routing of flows, based on their rates. This scheme assigns incoming flows to the queue associated to a specific traffic descriptor. q_0 is dedicated to process HP and MP flows, whereas q_1 is devoted to serve LP flows. Furthermore, the controller periodically checks the statistics related to the flows belonging to q_0 in order to figure out if a flow is not compliant to its constraint. When Beacon finds a MP flow which is violating its traffic descriptor, it re-routes it to the HR queue q_1 in order to be able to serve the traffic without causing congestion. If the newly re-routed flow overcomes a pre-defined threshold while traversing q_1, this traffic will be dropped by the Beacon controller. In the present simulation we set this threshold to 700kbit/s. We implemented this part of the strategy inside a specific Beacon module aimed at collecting statistic data.

We ran the performance analysis on a PC with Mininet (version 2.1.0) [Mininet]. The scenario is composed of two hosts connected to a SDN switch. The chosen implementation of the switch is Open vSwitch 2.0.2 [Ovs], managed by an instance of Beacon running on the same machine. Each port of the switch is configured with two queues, q_0 and q_1. We tested our strategy with a set of simulations involving 50 flows generated using *iperf*. Queue configuration and traffic characteristics are shown in Table 5.

Queue ID	Service Rate	Buffer Size
0	4 Mbit/s	1000 packets
1	16 Mbit/s	1000 packets

Traffic Descriptor	Rate	Percentage
HP/MP	40-60 kbit/s	40%
MP no conformant	200-800 kbit/s	20%
LP	200-800 kbit/s	40%

Table 5. Queues configuration and traffic characteristics.

This test is aimed at comparing the performances of our **Conformant** algorithm (as described before) with the **Dedicated** strategy. This last scheme consists in assigning each traffic class to the corresponding queue based on the traffic descriptor upon flow arrival and then take no further actions independently of the flow behaviour.

The results in Figure 23 show that, while the **Dedicated** strategy produces a 6.8% packet loss, the **Conformant** one allows to completely avoid the packet loss of High priority flows. This benefit is obtained together with the fact that the quality experienced by Low priority flows is not affected. It is worth noting that the loss of no conformant flows increases, but this is acceptable since these flows are not compliant with the constraints.

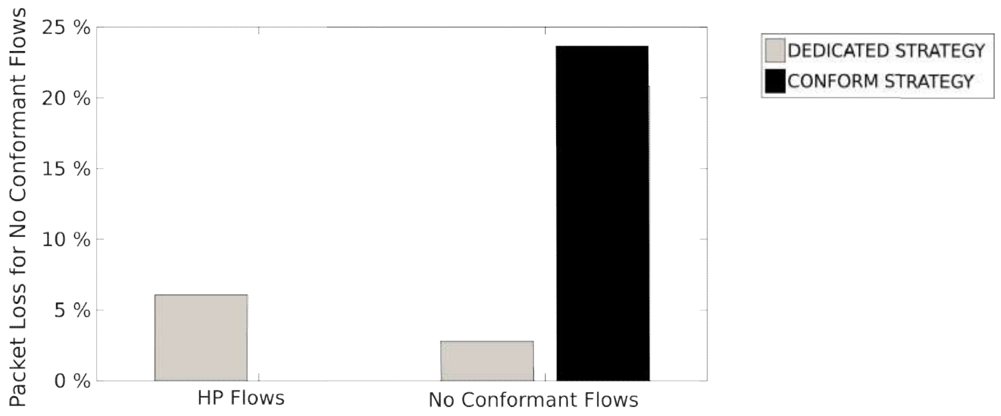


Figure 23. Conformant vs Dedicated Strategy.

Being our proposal a programmable solution, it is however possible to tune the threshold that defines the behaviour of the strategy in order to cope with different needs and situations. This parameter can be set through an external properties file, making the customization of the scheme even more flexible. In conclusion, we showed the results obtained in performance tests in which we compared the alternative QoS approaches. Our cases of study show that the proposed QoS solution allows getting good results when applied to the current OpenFlow environment. Future developments could consist in testing the network environment with a larger amount of traffic in order to test the scalability of our solutions. We also plan to devise alternative approaches such as exploiting the low rate queue in order to improve the quality perceived by high rate flows. Furthermore we plan to run our algorithms in other scenarios set with different queue configurations. Finally we hope to be able to conduct testbed measurements with commodity hardware routers in order to avoid the problems related to the software emulation of this type of devices.

VI. Incremental Deployment of STS-SDN solution

In previous sections we have seen how the SDN paradigm can improve networks in terms of reliability and programmability. Software Defined Network can also address many of the requirements of a next-generation STS because it offers a way to deploy a network that is flexible, fault-tolerant and QoS-aware. Since Smart Transportation Systems are typically deployed in large cities with dense population and involve different actors at different levels as said before, we cannot think to build an ad-hoc network to handle all STS generated traffic, but we need to use already-existent networks and services offered by a set of operators/service providers. Nowadays, service providers and network operators do not have SDN-compliant networks, but they are planning to evolve their infrastructures towards this new paradigm.

One of the major problems in SDN is that deploying a SDN network means to change the entire network structure. This can be easily done in relatively small environments such as data-centers and campuses, where we can already find a variety of deployments, such as Google Andromeda [GoogleAndromeda] used for the Google's data centers or

OpenStack Neutron [OpenStackNeutron] used by Rackspace's cloud services. However, from the network operator point of view, the process to adopt SDN in its infrastructure is very costly, because a very large amount of devices at different levels are involved in this change. Furthermore, an operator have to train its programmers and personnel to interact with such new paradigm, and this requires investments in terms of money and time. For all these reasons, operators could be reluctant to do this upgrade. A possible solution that has been considered by large-scale operators, such as AT&T [ATT], is an incremental deployment of SDN into the already-existent infrastructure, since SDN nodes can be configured to be transparent to the other "legacy" nodes. In this way, deployment can be done without affecting network functionalities and preserving network access. Another advantage of the incremental deployment is that such new functionalities can be tested and evaluated before making major changes to the infrastructure. Of course, with this method, the time-to-market of the overall network is considerably higher but this problem is covered by the other advantages. Doing this incremental deployment, we introduce a sort of hybrid SDN network where traditional forwarding methods (called Comercial Off-The-Shelf Networks or CN) and newest one coexist.

As shown in [Vissicchio14], we can have different types of hybrid SDN networks:

- Topology-based: in this model we have a topological separation between SDN and CN with adapters to make the different zones to interact. This kind of deployment introduces SDN at regional-level, and could be adopted if we want to extend SDN starting from particular kind of zones (e.g. large cities). Operators can take advantage from this model by means of tests and by isolating major failures on their new deployments.
- Service-based: in this model we use SDN only for a subset of network services and forwarding types, leaving CN to handle all the others. For example, we could use SDN at the edge nodes to improve load balancing and traffic engineering and leave all core-network functionalities to CN. With this model we can strategically place SDN nodes in the network and incrementally deploy new services as they will be ready to be handled by SDN.
- Traffic-based: in this model SDN handles only a certain class of traffic, while CN handles the other ones. In this way we can initially forward by SDN only the lowest priority traffic, and incrementally switch the other traffic classes from CN to SDN where the model is better consolidated. With traffic-based model we need to have many SDN-compliant nodes (e.g. nodes that are OpenFlow capable), since we put both CN and SDN paradigms in all the nodes of the network.
- Integrated model: in this model there are no SDN nodes, but SDN-like working is obtained by controlling the CN nodes and transferring the control plane at the controller node. In this way the behaviour of all distributed routing protocols such as BGP, OSPF, etc. is managed directly by the controller that sends to CN nodes all the messages to create, for example, a particular forwarding path or other SDN-like behaviours. This method has a clear advantage: no SDN nodes are required and so this kind of hybrid model can be easily deployed into operators' networks. Obviously this solution offers a SDN-like network where we cannot have all innovative functionalities that are proper of such paradigm, and the complexity of the controller could be not negligible. The choice of the solution depends on different factors. First of all operators need to decide where to make the first changes and define a sort of "road-map" of the next steps. Doing this is not a trivial process, because operators have to decide on which level of the network to do the deployment considering

that each level has different peculiarities, with obvious impact over network performance and deployment cost.

Summarizing, operators must take into account, for each kind of intervention:

- the number of involved devices;
- the cost of the devices substitution/update;
- the benefits to have an SDN node in such place;
- the interaction between the "legacy" nodes.

With these parameters operators can define an effective incremental deployment of the SDN-compliant network, choosing the proper model that fits its needs and requirements.

VII. Conclusions

In this chapter we have analysed the emerging technologies in the fields of the Smart Cities, focusing on Smart Transportation Systems. After that, we have discussed about the possible infrastructures on which STS relies, founding that telecommunications networks are one of the most critical aspects because emergency conditions in the cities (hurricane, hearthquakes, etc.) could damage the network (or part of it) resulting in interruption of service that, for a STS, may result in putting citizens and city infrastructures in danger. Facing with network survivability we have seen that current network paradigms and protocols are not able to guarantee sufficient level of service in case of disaster, and we saw how a new emergent network paradigm such as SDN could cope with this problems. Furthermore, we have proposed a simple example on how an application in a STS environment could take advantage from the SDN paradigm in terms of Quality of Service. Finally, we have considered how to deploy the SDN solution over operator networks and discussed related issues and open questions.

VIII. References

[Afanasyev10] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock. 2010. Host-to-Host Congestion Control for TCP. *Commun. Surveys Tuts.* 12, 3 (July 2010), 304-342.

[Agarwal13] S. Agarwal, M. Kodialam, and T. Lakshman. Traffic engineering in software defined networks. In INFOCOM, 2013 Proceedings IEEE , pages 2211-2219, April 2013.

[Alcatel]: "The IoT: The next step in internet evolution", available at http://www2.alcatel-lucent.com/techzine/iot-internet-of-things-next-step-evolution/

[Applegate03] D. Applegate and M. Thorup. Load optimal mpls routing with n + m labels. In INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies, volume 1, pages 555-565, March 2003

[Aurrecoechea98] C. Aurrecoechea, A. T. Campbell, and L. Hauw, "A survey of qos architectures," Multimedia Syst., vol. 6, pp. 138 – 151, May 1998.

[Bhattacharjee97] Bhattacharjee, S., Calvert, K.L.., Zegura, E. W. 1997. An architecture for active networks. In *Proceedings of High-Performance Networking*.

[Bird] BIRD Internet routing daemon; http://bird.network.cz/

[Boero15a] L. Boero, M. Cello, C. Garibotto, M. Marchese, M. Mongelli, "Management of Non-Conformant Traffic in OpenFlow Environments", International Symposium on Performance Evaluation of Computer and Telecommunication Systems 2015, SPECTS 2015, Chicago, USA.

[Boero15b] L. Boero, M. Cello, C. Garibotto, M. Marchese, M. Mongelli, "BeaQoS: Quality of Service Support in OpenFlow", draft.

[Caesar05] Caesar, M., Feamster, N., Rexford, J., Shaikh, A., van der Merwe, J. 2005. Design and implementation of a routing control platform. In *Proceedings of the 2nd Usenix Symposium on Networked Systems Design and Implementation (NSDI).*

[Campbell94] A. Campbell, G. Coulson, and D. Hutchison, "A quality of service architecture," SIGCOMM Comput. Commun. Rev., vol. 24, pp. 6 – 27, April 1994.

[Casado07] Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., Shenker, S. 2007. Ethane: taking control of the enterprise. In *Proceedings of ACM SIGCOMM*.

[Chourabi14]: H. Chourabi, T. Nam, S. Walker, J. R. Gil-Garcia, S. Mellouli, K. Nahon, T. A. Pardo, H. J. Scholl. Understanding Smart Cities: An Integrative Framework. 2014 45th Hawaii International Conference on System Sciences.

[Erickson13] D. Erickson. The Beacon Openflow Controller. In Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, pages 13-18, 2013

[Ezell10]: ITIF The Information Technology & Innovation Foundation, "Intelligent Transportation Systems", Stephen Ezell, January 2010.

[Farrel06] Farrel, A., Vasseur, J.,-P. Ash, J. 2006. A Path Computation Element (PCE)-based architecture. Internet Engineering Task Force RFC 4655. https://tools.ietf.org/html/rfc4655

[Feamster04] Feamster, N., Balakrishnan, H., Rexford, J., Shaikh, A., van der Merwe, K. 2004. The case for separating routing from routers. In Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture.

[Feamster14] Nick Feamster, Jennifer Rexford, and Ellen Zegura. 2014. The road to SDN: an intellectual history of programmable networks. SIGCOMM Comput. Commun. Rev. 44, 2 (April 2014), 87-98.

[Filsfils2015] C. Filsfils, S. Previdi, B. Decraene, S. Litkowski, R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-03, 2015, https://www.ietf.org/id/draft-ietf-spring-segment-routing-03.txt

[Genova]: Genova Smart City website: http://www.genovasmartcity.it

[GoogleAndromeda]: “Enter the Andromeda zone - Google Cloud Platform’s latest networking stack”. Available at http://googlecloudplatform.blogspot.it/2014/04/enter-andromeda-zone-google-cloud-platforms-latest-networking-stack.html

[Goransson14] Paul Goransson and Chuck Black. 2014. *Software Defined Networks: A Comprehensive Approach* (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Greenberg05] Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., Zhang, H. 2005. A clean-slate 4D approach to network control and management. *ACM SIGCOMM Computer Communication Review* 35(5): 41-54.

[Gude08] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S. 2008. NOX: Towards an operating system for networks. *ACM SIGCOMM Computer Communication Review* 38(3): 105-110.

[Handley05] Handley, M., Kohler, E., Ghosh, A., Hodson, O., Radoslavov, P. 2005. Designing extensible IP router software. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)*.

[Hong03] D. W.-K. Hong and C. S. Hong, “A qos management framework for distributed multimedia systems,” Int. J. Netw. Manag., vol. 13, pp. 115 – 127, March 2003.

[Ibm]: IBM, Cisco and the business of smart cities - How two of the IT industry's largest companies plan to rewire urban living. Available at http://www.information-age.com/industry/hardware/2087993/ibm-cisco-and-the-business-of-smart-cities

[Jain13] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., Vahdat, A. 2013. B4: experience with a globally deployed software-defined WAN. In *Proceedings of ACM SIGCOMM*.

[Katz03] D. Katz, K. Kompella, D. Yeung, “Traffic Engineering (TE) Extensions to OSPF Version 2”, RFC3630, September 2003, https://tools.ietf.org/html/rfc3630

[Lakshman04] Lakshman, T. V., Nandagopal, T., Ramjee, R., Sabnani, K., Woo, T. 2004. The SoftRouter architecture. In Proceedings of the 3rd ACM Workshop on Hot Topics in Networks (HotNets); http://conferences.sigcomm.org/hotnets/2004/HotNets-III%20Proceedings/lakshman.pdf.

[Lopez1]: Lopez Research, “Smart Cities Are Built On The Internet Of Things”

[Marchese07] M. Marchese, QoS Over Heterogeneous Networks. Wiley, 2007.

[McKeown08] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J. 2008. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38(2): 69-74.

[McKeown11] Nick McKeown (Stanford), ITC Keynote, San Francisco, 2011. http://yuba.stanford.edu/~nickm/talks/ITC%20Keynote%20Sept%202011.ppt

[MIlano]: Milano Smart City website: http://www.milanosmartcity.org

[Nicira12] Nicira. It's time to virtualize the network. 2012; http://www.netfos.com.tw/PDF/Nicira/It%20is%20Time%20To%20Virtualize%20the%20Network%20White%20Paper.pdf.

[Nunes14] Nunes, B.A.A.; Mendonca, M.; Xuan-Nam Nguyen; Obraczka, K.; Turletti, T., "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *Communications Surveys & Tutorials, IEEE* , vol.16, no.3, pp.1617,1634, Third Quarter 2014

[OpenStackNeutron]: OpenStack Neutron description. Available at https://wiki.openstack.org/wiki/Neutron

[Openflow15] Open Networking Foundation, "OpenFlow Switch Specification", Version 1.5.1, March 26, 2015. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf

[Quagga] Quagga software routing suite; http://www.quagga.net/.

[Salim03] Salim, J., Khosravi, H., Kleen, A., Kuznetsov, A. 2003. Linux Netlink as an IP services protocol. Internet Engineering Task Force, RFC 3549. https://tools.ietf.org/html/rfc3549.

[Saltzer81] Saltzer, J. H., D. P. Reed, and D. D. Clark (1981) "End-to-End Arguments in System Design". In: Proceedings of the Second International Conference on Distributed Computing Systems. Paris, France. April 8–10, 1981. IEEE Computer Society, pp. 509-512.

[ATT]: "SDN and NFV will come to life in the operator network, eventually". Available at http://searchsdn.techtarget.com/news/2240215704/SDN-and-NFV-will-come-to-life-in-the-operator-network-eventually.

[Shenker11] Scott Shenker (UC Berkeley), "The Future of Networking, and the Past of Protocols", Open Network Summit, 2011. http://www.opennetsummit.org/archives/oct11/shenker-tue.pdf

[Shenker12] Scott Shenker (UC Berkeley), "A Gentle Introduction to Software Defined Networks", Technion Computer Engineering Center, 2012. http://tce.technion.ac.il/files/2012/06/Scott-shenker.pdf

[Shenker13] Scott Shenker (UC Berkeley), "Software-Defined Networking at the Crossroads", Standford, Colloquium on Computer Systems Seminar Series (EE380), 2013.

[Smith96] Smith, J., et al. 1996. SwitchWare: accelerating network evolution. Technical Report MS-CIS-96-38, University of Pennsylvania.

[Sterbenz10]: J. P.G. Sterbenz, D. Hutchison, E. K. Çetinkaya, A.Jabbar, J. P. Rohrer, MSchöller, P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines". Computer Networks Volume 54, Issue 8, 1 June 2010, Pages 1245–1265, Elsevier.

[Torino]: Torino Smart City website: http://www.torinosmartcity.it [vanderMerwe06] van der Merwe, J., Cepleanu, A., D'Souza, K., Freeman, B., Greenberg, A., et al. 2006. Dynamic connectivity management with an intelligent route service control point. In *ACM SIGCOMM Workshop on Internet Network Management.*

[Vissicchio14] Stefano Vissicchio, Laurent Vanbever, and Olivier Bonaventure. 2014. Opportunities and research challenges of hybrid software defined networks. *SIGCOMM Comput. Commun. Rev.* 44, 2 (April 2014), 70-75.

[Wetherall98] Wetherall, D., Guttag, J., Tennenhouse, D. 1998. ANTS: a toolkit for building and dynamically deploying network protocols. In *Proceedings of IEEE OpenArch*.

[Wu82] Chuan-lin Wu; Tse-Yun Feng; Min-Chang Lin, "Star: A Local Network System for Real-Time Management of Imagery Data," *Computers, IEEE Transactions on* , vol.C-31, no.10, pp.923,933, Oct. 1982.

[Yang04] Yang, L., Dantu, R., Anderson, T., Gopal, R. 2004. Forwarding and Control Element Separation (ForCES) Framework. Internet Engineering Task Force, RFC 3746. https://www.rfc-editor.org/rfc/rfc3746.txt

[Lta14] http://www.lta.gov.sg/content/ltaweb/en/roads-and-motoring/managing-traffic-and-congestion/intelligent-transport-systems/SmartMobility2030.html

[Etsi] http://www.etsi.org/technologies-clusters/technologies/intelligent-transport