FireDeX: a Prioritized IoT Data Exchange Middleware for Emergency Response

Georgios Bouloukakis[1,2]

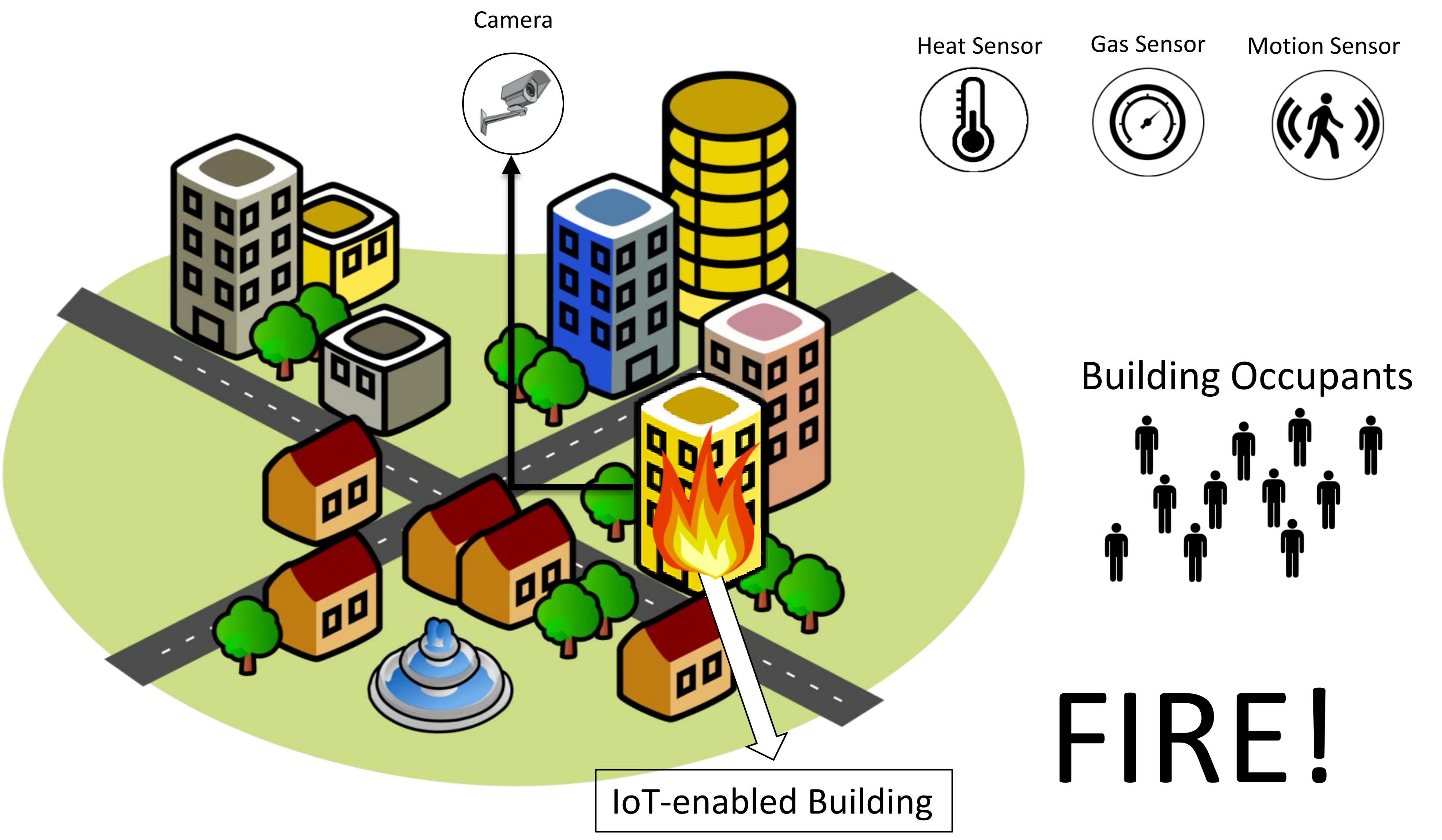
Rennes, France, December 2018

2018 ACM/IFIP International Middleware Conference

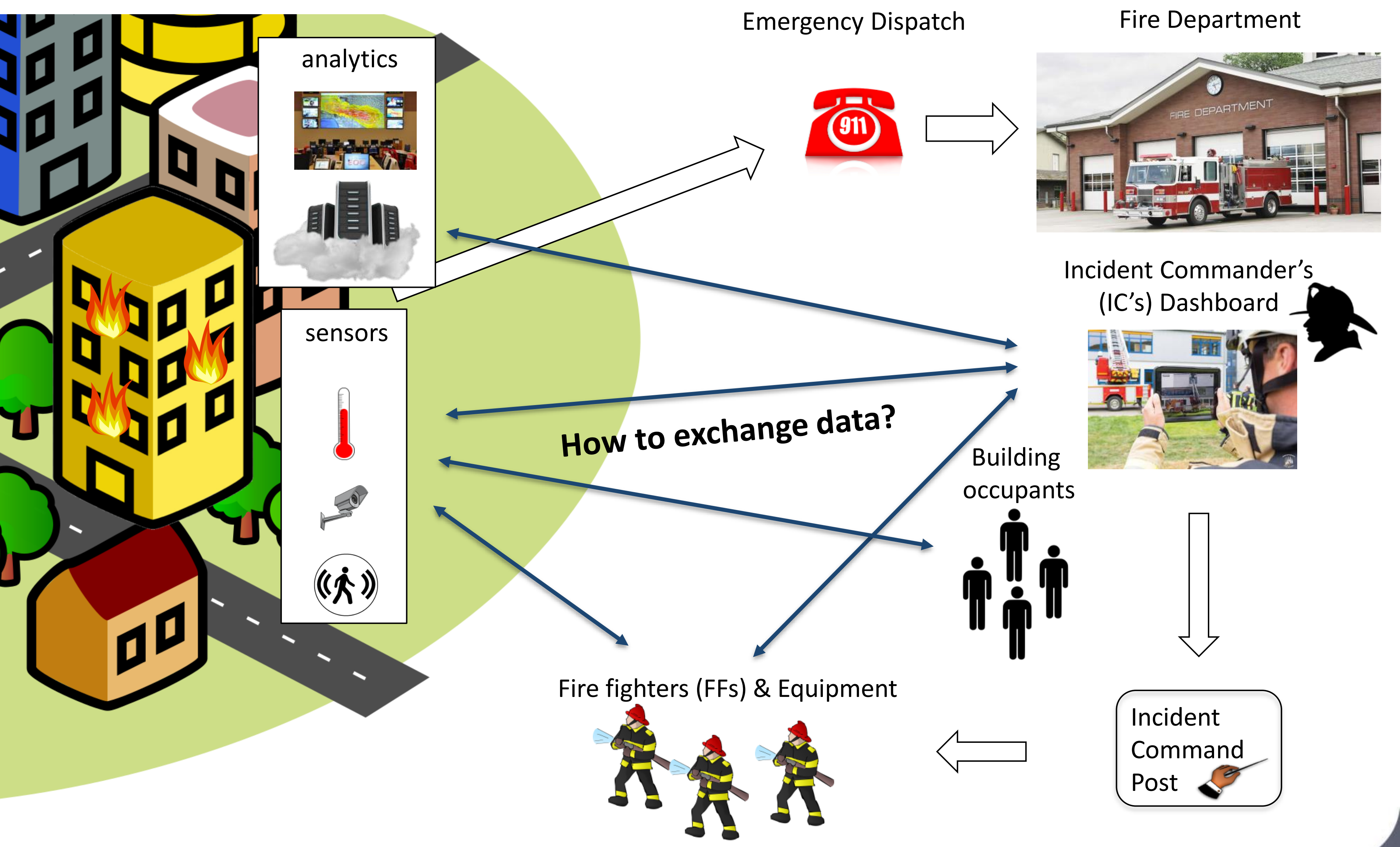
Joint work with Kyle Benson[1], Casey Grant[3], Valérie Issarny[2], Sharad Mehrotra[1], Ioannis Moscholios[4], Nalini Venkatasubramanian[1]

[1]Donald Bren School of Information and Computer Sciences, UC Irvine, USA
[2]MiMove team, Inria Paris, France
[3]National Fire Protection Association, USA
[4]Dept. of Informatics & Telecommunications, Univ. of Peloponnese, Greece

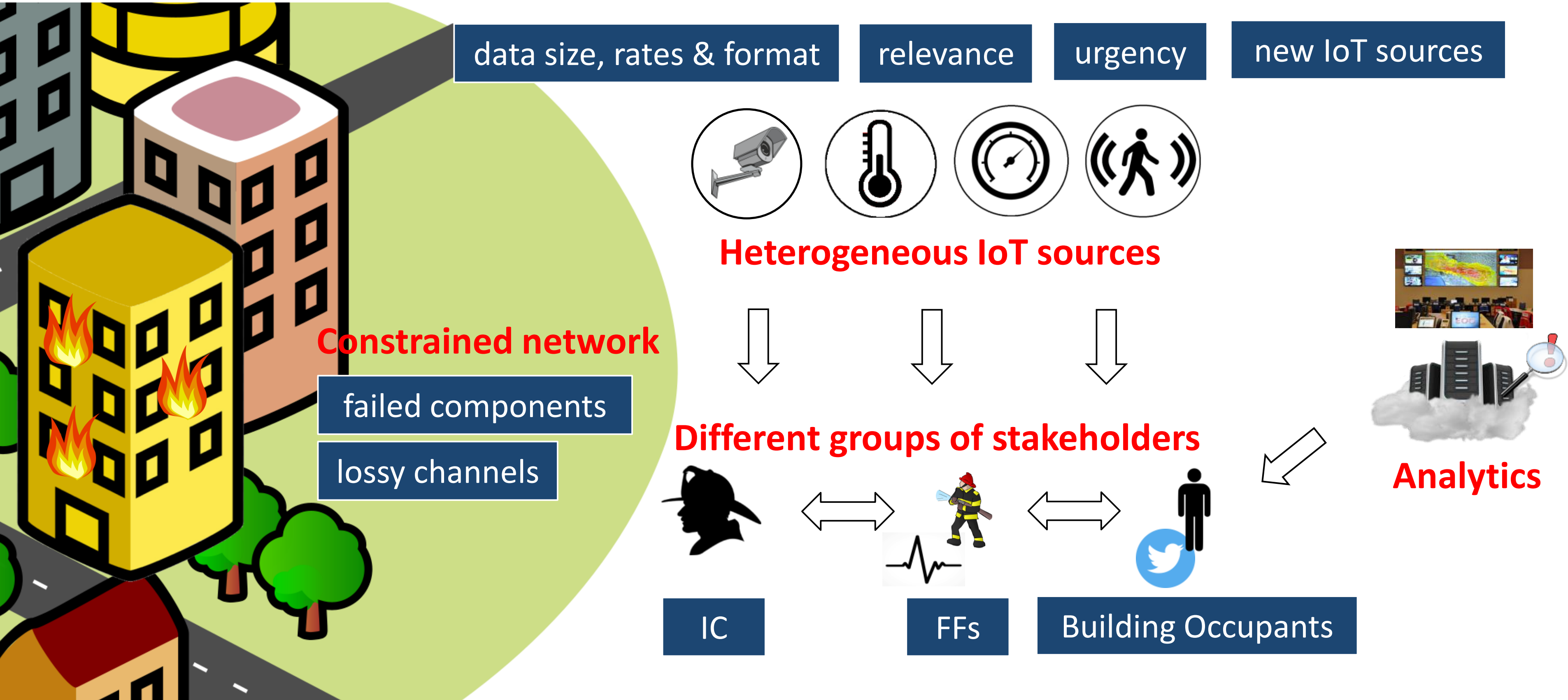
Motivation: IoT-enhanced structural fire response



Motivation: IoT-enhanced structural fire response



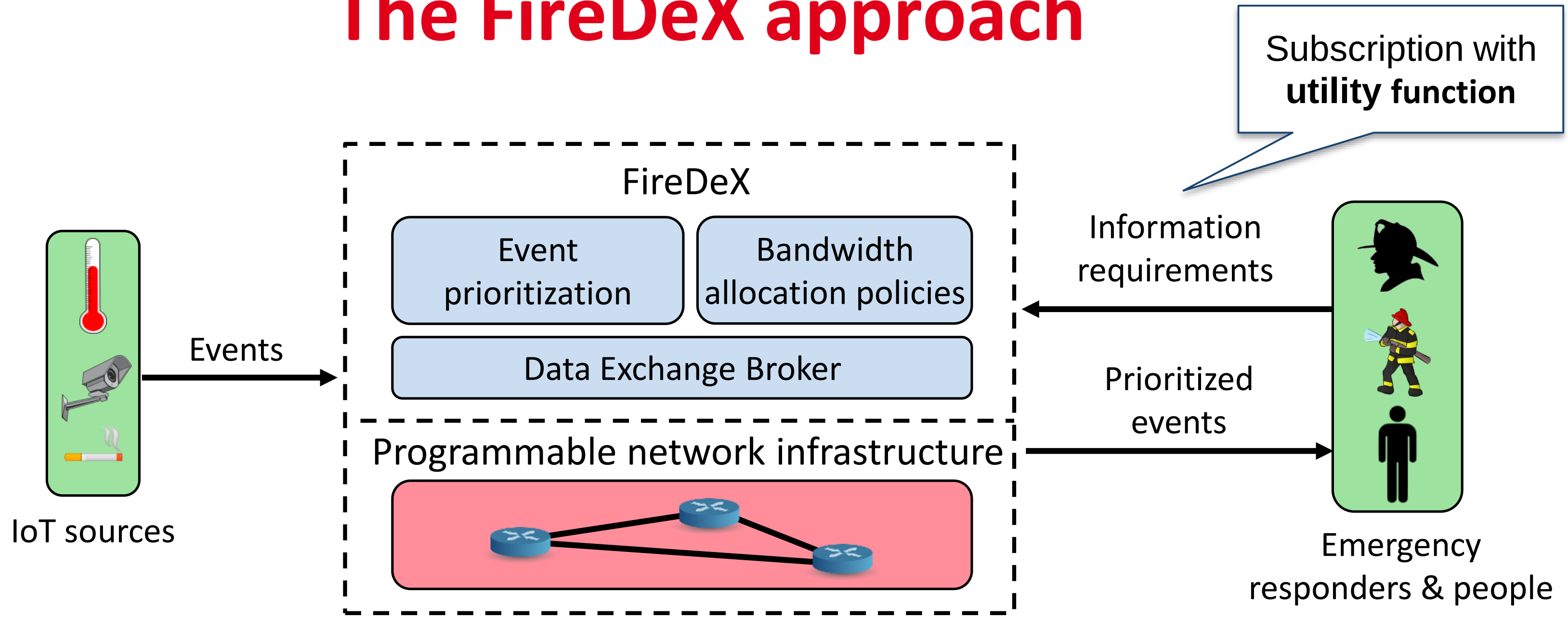
Motivation: IoT-enhanced structural fire response



- **Problem**: how to enable the exchange of **heterogeneous data** by taking into account stakeholders' **information requirements** and **network conditions** as a scenario evolves?

FireDeX

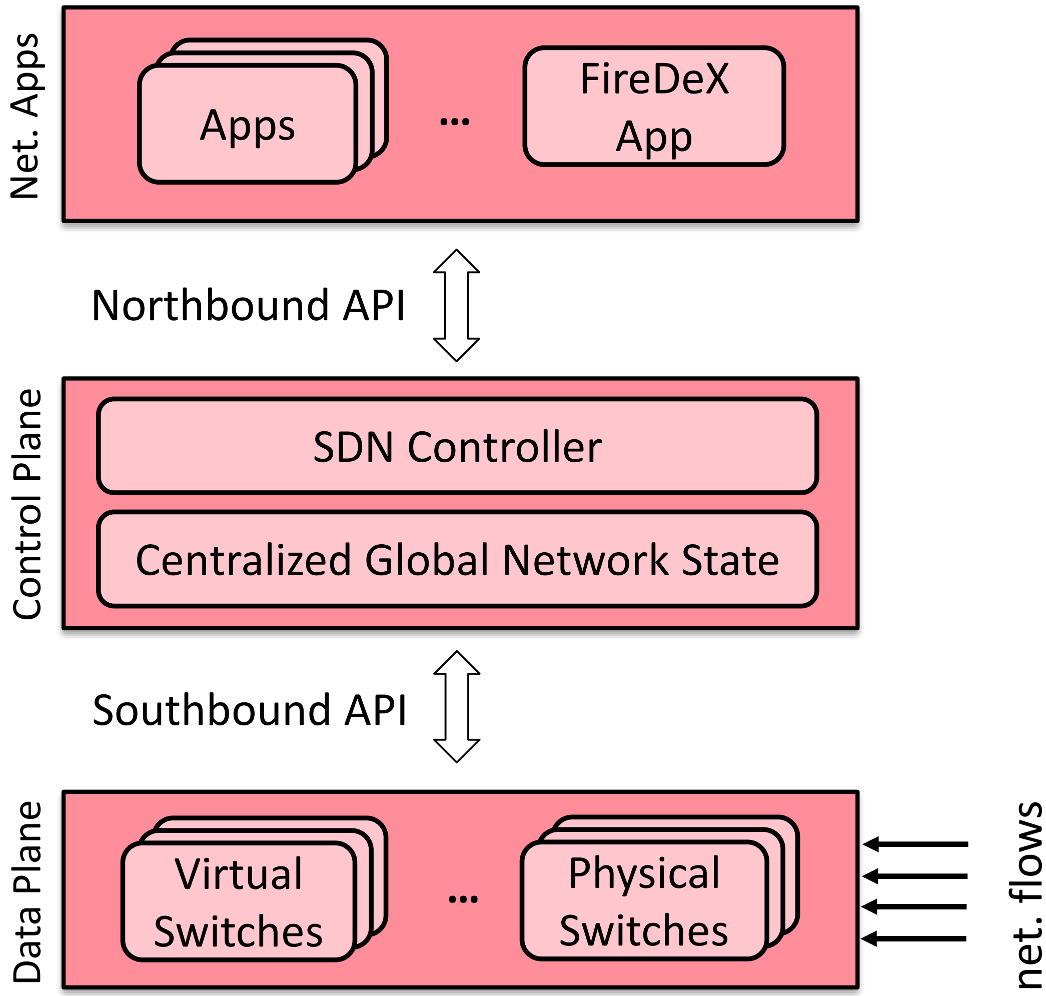
The FireDeX approach



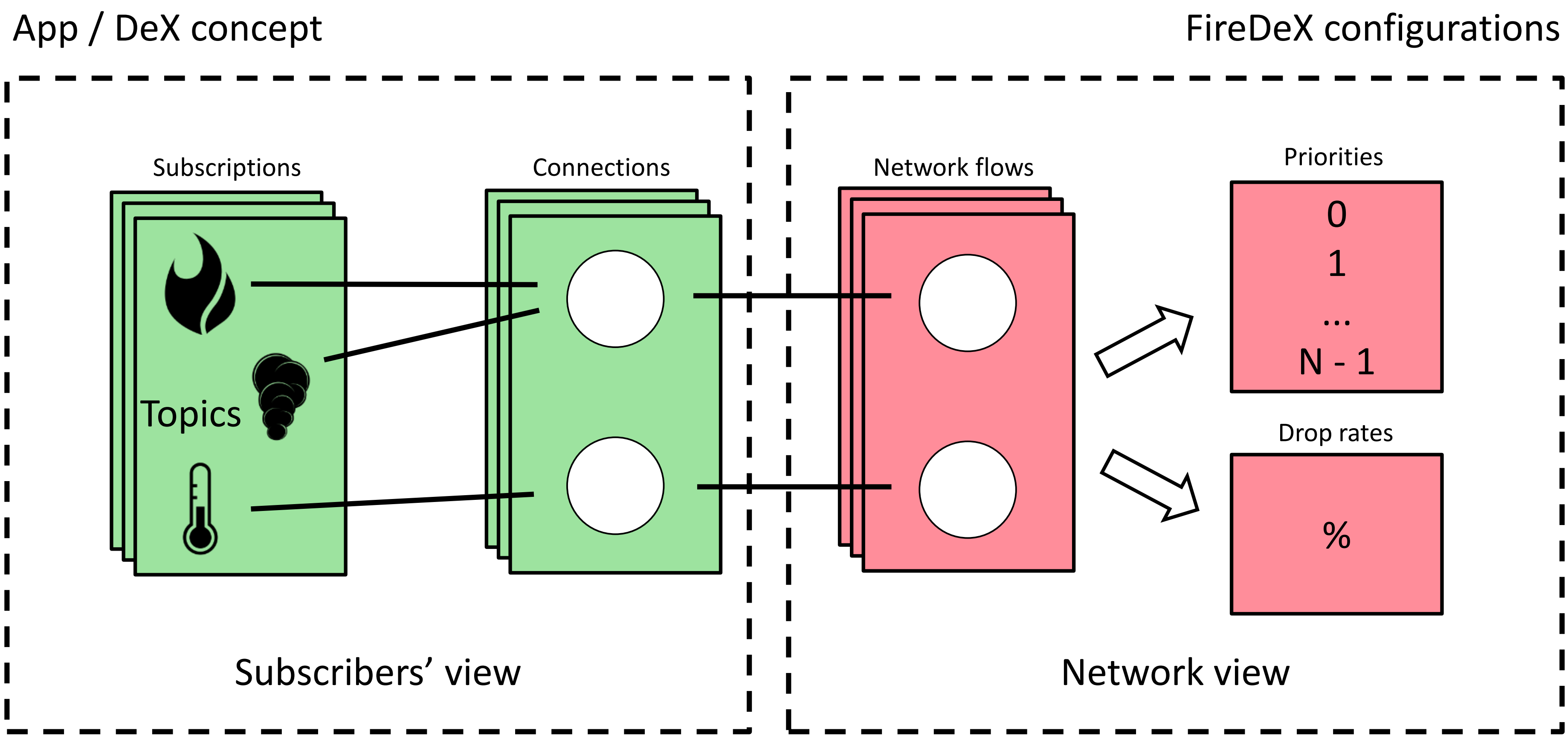
- FireDeX middleware configures the data exchange & network with prioritization and bandwidth allocation policies based on:
  - information requirements
  - network resource constraints
- Rely on SDN to bridge critical information requirements with network flows.
- Model the performance of FireDeX across multiple layers using Queueing Theory.
- Use the underpinning formal model for deriving novel algorithms that prioritize IoT events and tune notification delivery/response times.

Goal: timely and reliable delivery of the most critical data to relevant subscribers despite challenging network conditions.

SDN-background

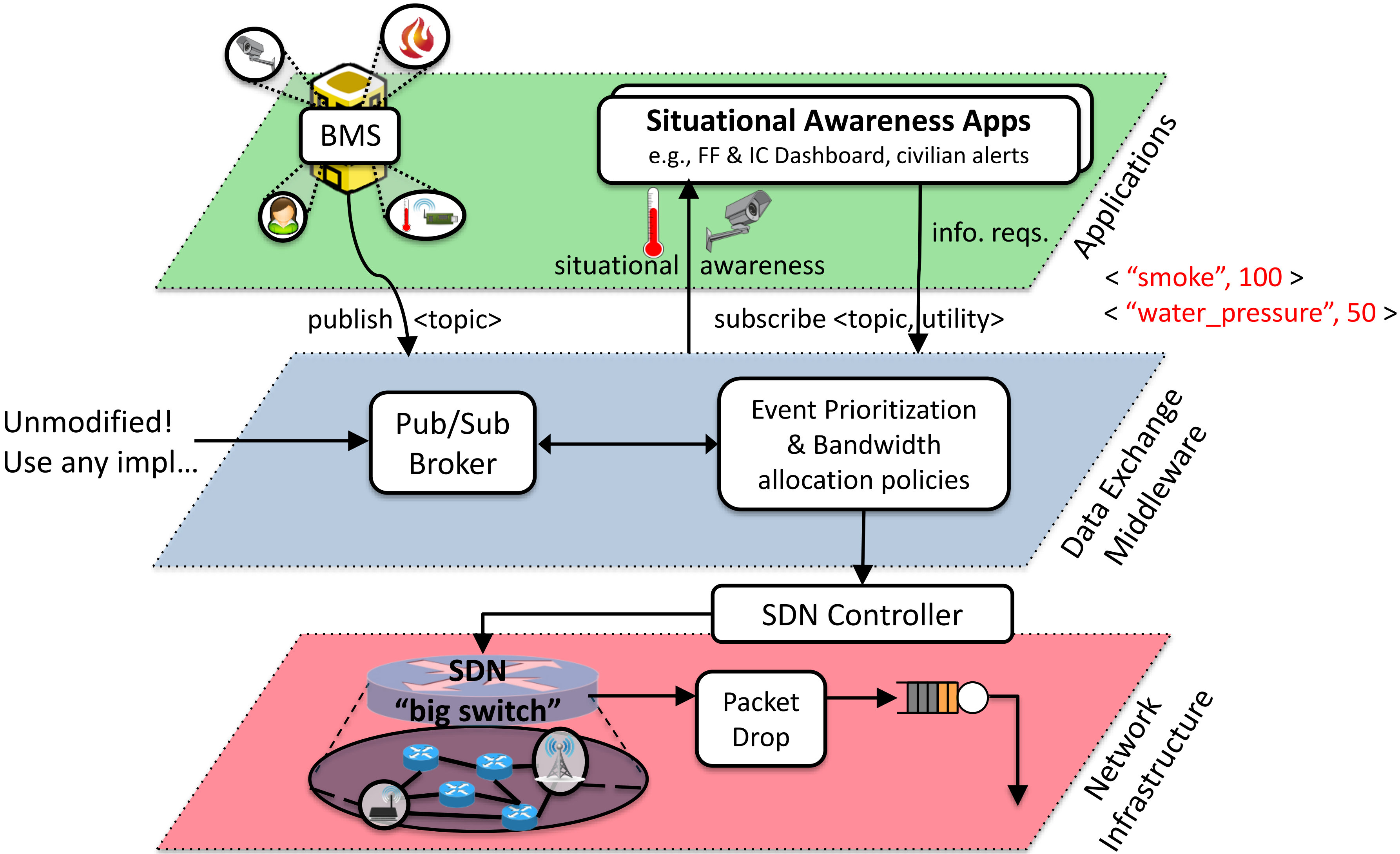


Mapping info. reqs. to network state

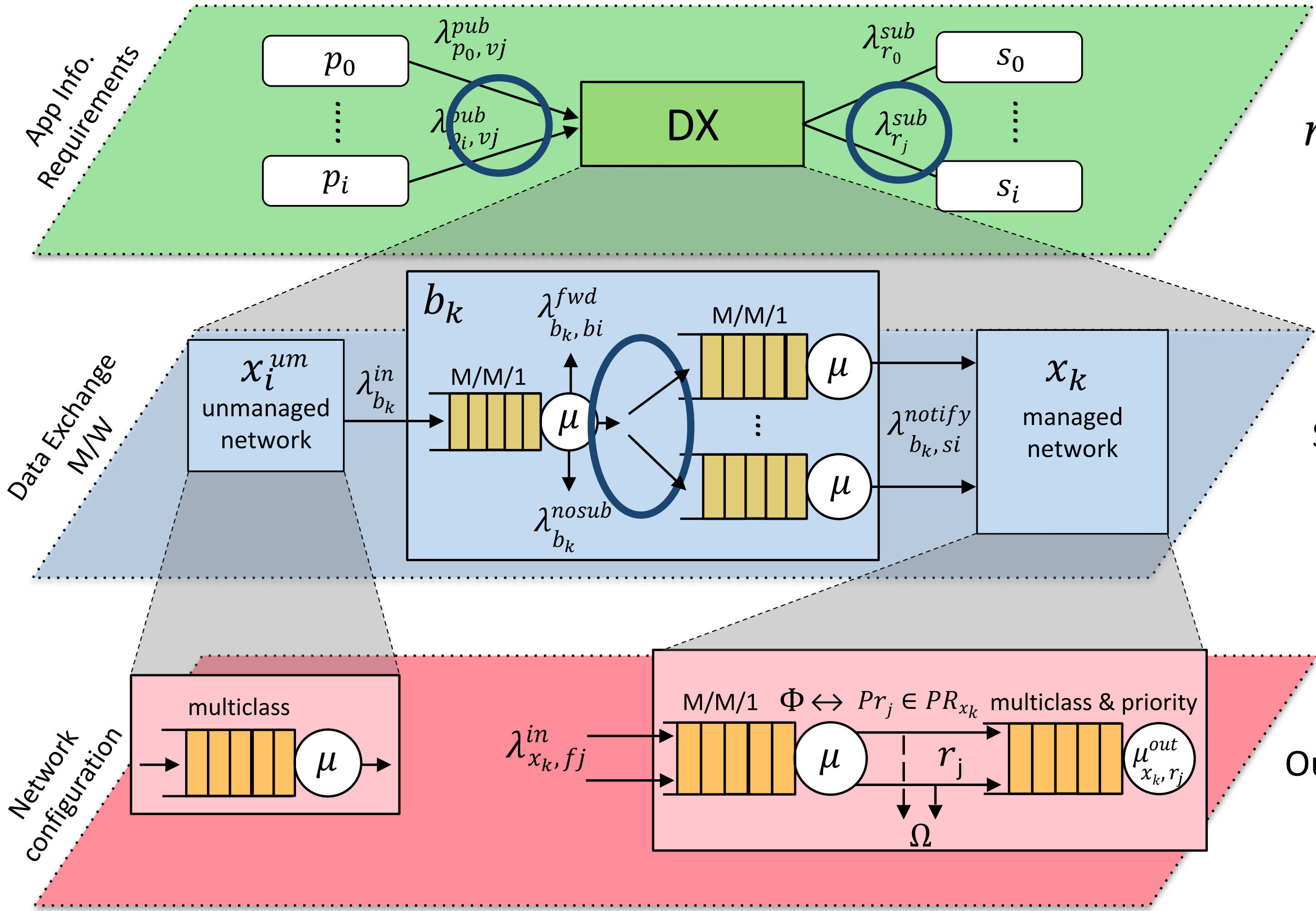


- Network flows enable SDN infrastructure to differentiate subscriptions (e.g. by UDP/TCP port number + IP addr).

FireDeX across layers



Modeling FireDeX using queuing theory



$r_j = (s_i, v_j, U_{r_j})$

Subscription matching

Our new queueing model

Prioritization algorithm

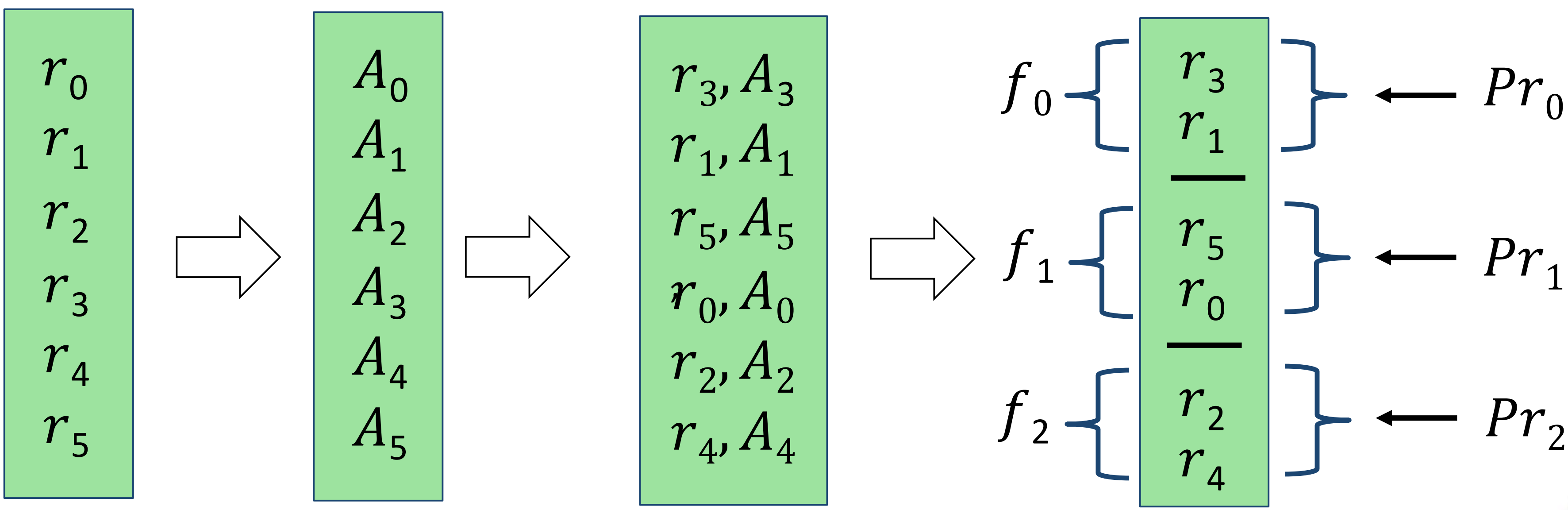
1. Estimate the adjusted utility function per subscription: information value per unit of *bandwidth*.
2. Sort subscriptions.
3. Group them into approximately equal-sized network flows.
4. Priorities assigned to approximately equal-sized groups of network flows.

Maximum utility achievable for subscription r_j

$$A_{r_j} = \frac{\hat{U}_{r_j}}{G_{v_j}\lambda^{notify}_{b_k,r_j}}$$

Serialized packet size for topic v_j notifications

Rate of notifications (publications) for subscription r_j



Drop rate algorithms

Flat

Drop rates for each network flow

$$\Omega(f_j) = \beta$$

Linear

$$\Omega(f_j) = \beta\Phi(f_j)$$

Mapped priority to network flow

Exponential

$$\Omega(f_j) = 1 - \beta^{-\Phi(f_j)}$$

Optimized

1. Formulated as a convex optimization problem.
   - Maximize overall utility as sum of all subscriptions' utilities.
   - Enabled by choice of logarithm for utility function.
2. 2nd constraint: queue stability condition.
   - Ensures allocated bandwidth within that available.

$$\begin{aligned} \text{maximize} \quad & \sum_{r_j \in R} \alpha_{r_j} \log(1 + \Xi_{r_j}) \\ \text{subject to} \quad & \Omega(f_j) \in [0, 1], \forall f_j \in F \\ & \sum_{r_j \in R_{x_k}} \frac{\lambda^{thru}_{x_k, r_j}}{\mu^{out}_{x_k, r_j}} \le 1 - \tilde{\rho}, \forall x_k \in X \end{aligned}$$

"Rho tolerance" enables keeping a buffer within the bandwidth (~0.1)

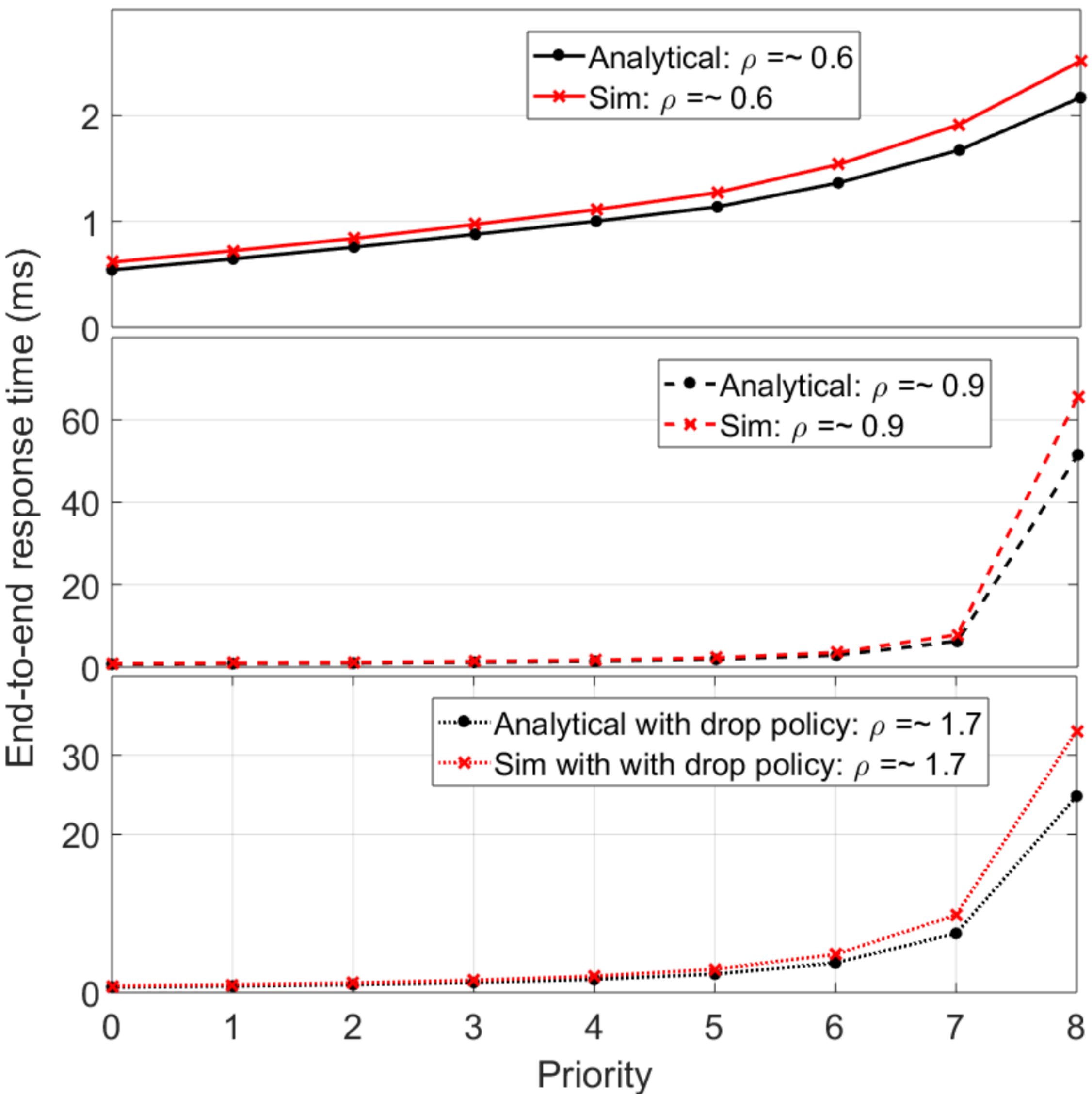
Experimental setup

- We validate our theoretical model, evaluate the FireDeX approach and compare different prioritization and dropping algorithms.
- We use JINQS (Java Implementation of a Network-of-Queues Simulation) to build our queueing network: an open source simulator for building queueing networks.
  - We have extended JINQS to implement our new multiclass priority queueing model.

Data Exchange Parameters					
	#topics ($\vert V\vert$)	**pub rate** ($\lambda^{pub}_{p_i,v_j}$)	**event size** (G_{v_j})	**#subscriptions** ($\vert R_{s_i}\vert$)	**utility weight** (α_{r_j})
Telemetry data	140	$Exp(\frac{1}{6}) \in [4,7]$	$Exp(\frac{1}{110}) \in [90,500]$	70	$Exp(\frac{1}{0.5}) \in [0.01,2]$
Async. events	60	$Exp(\frac{1}{4}) \in [3,5]$	$Exp(\frac{1}{800}) \in [500,1100]$	42	$Exp(1) \in [0.1,4]$

Network Parameters					
#subscribers ($\vert S\vert$)	**#publishers** ($\vert P\vert$)	**#flows** ($\vert F_{s_i}\vert$)	**#priorities** ($\vert Y\vert$)	**bandwidth** (w_{s_i})	ρ **tolerance** ($\tilde{\rho}$)
10	160	9	9	80 Mbps	0.1

Model validation: varying traffic loads



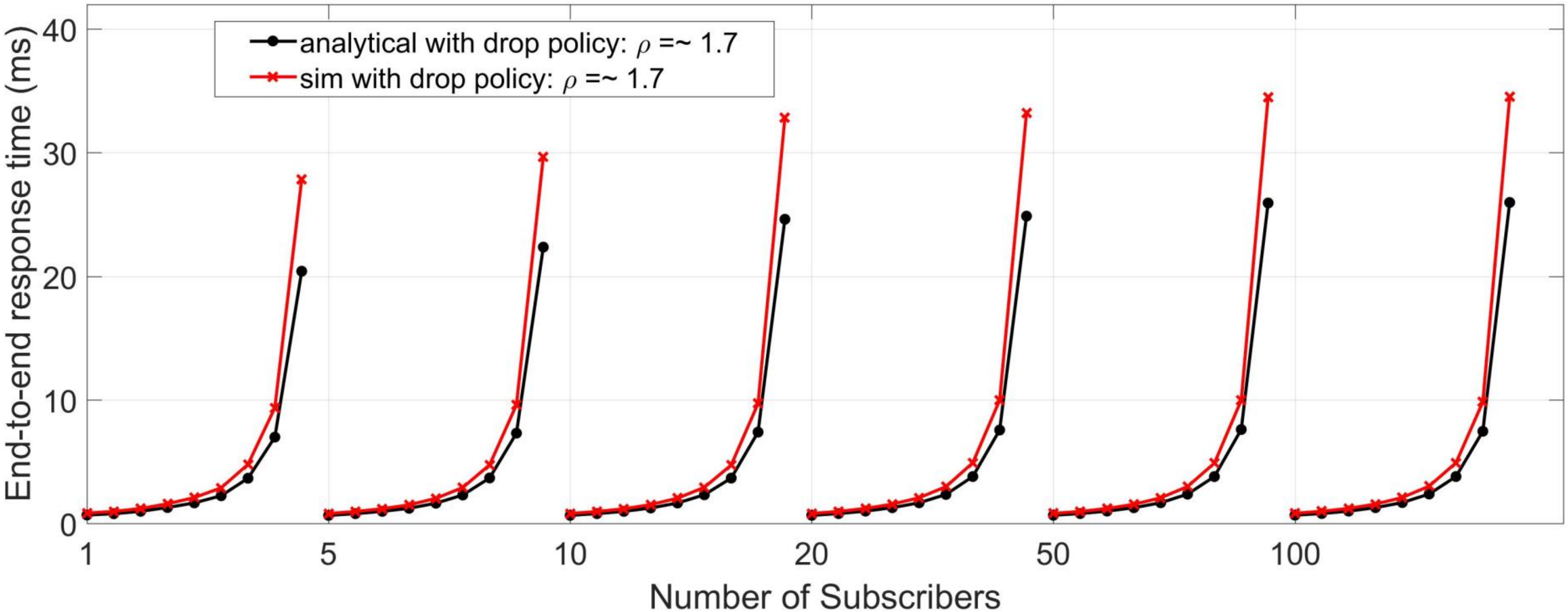
→ Under-loaded

→ Saturated

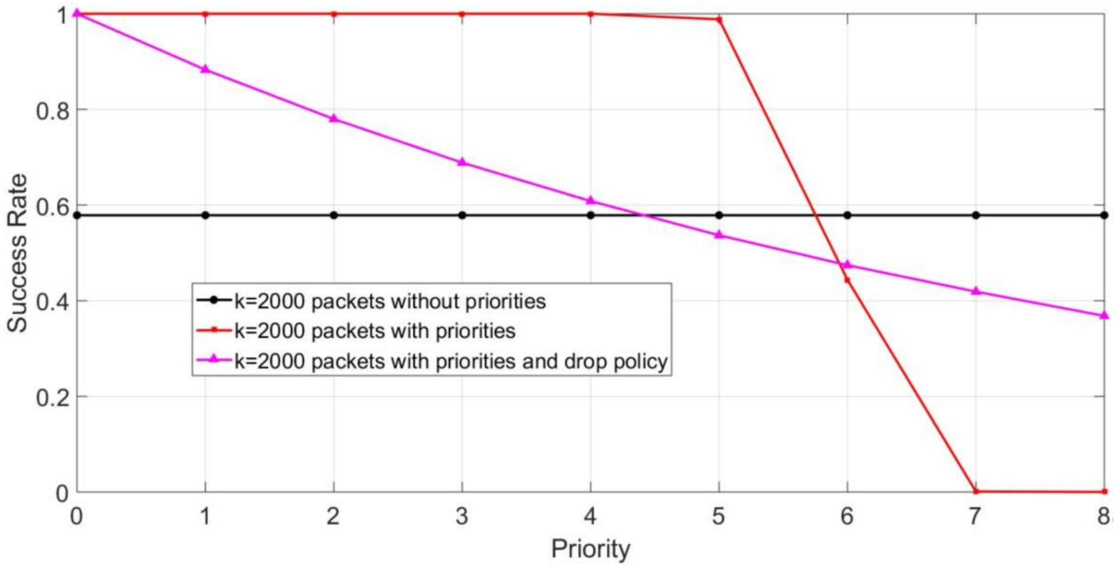
- Analytical model for lowest priorities is slightly less accurate.

→ Overloaded

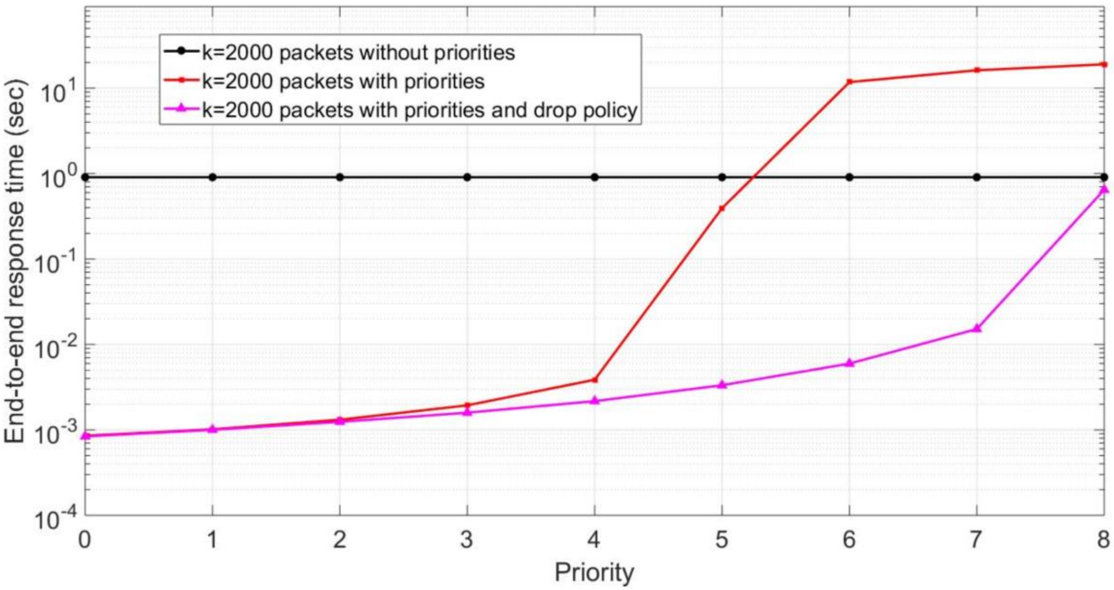
Model validation: scalability



FireDeX approach evaluation

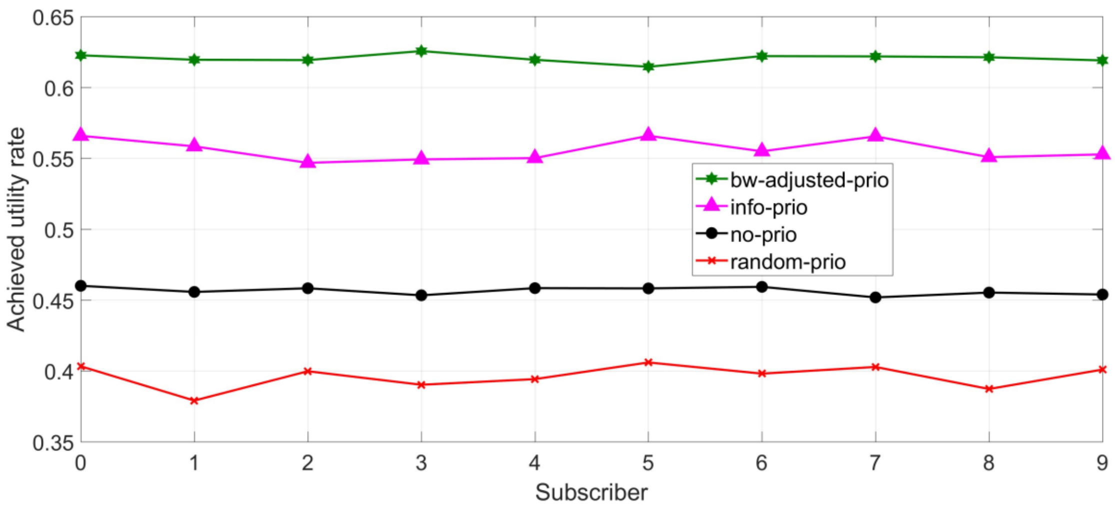


- With an **overloaded** system, switch **buffers fill up** and cause high delay / packet drops.
- Our approach **delivers more** high priority events than finite buffers only.



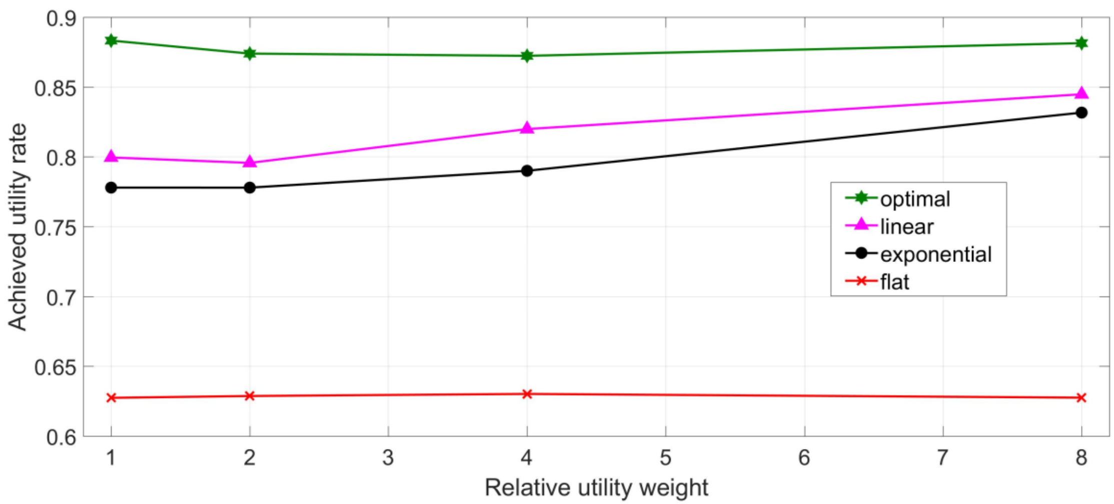
- High priority events also **delivered quicker**.
- Addition of **drop rate** policy smooths success rate while reducing end-to-end delay.

Algorithms comparison



Prioritization algorithms:

- Our **bandwidth-aware** greedy strategy performs better than bandwidth-unaware version.
- Both better than **no prioritization**.
- But **random** priorities are worst: need to set priorities correctly!



Drop rate algorithms

- **Convex optimization** performs best in comparison to linear, exponential and flat policies (drop rates by **assigned priority**).
- Plot shows **varying utilities** of async events vs. data telemetry: simpler policies perform closer to optimal for larger differences.

Conclusions & Next steps

Conclusions

- We introduce a middleware that integrates application and network awareness.
- Our application-aware **prioritization algorithm** improves the value of exchanged information by 36% when compared with no prioritization.
- Network-aware **drop rate policies** improve this performance by 42% over **priorities only** and by 94% over **no prioritization**.

Next steps

Queueing model:

- Consider non-Poisson arrival and service rates by using G/G/1 or G/D/1 queues.

System:

- Alternative utility functions.
- Tuning the entire broker network.
- Use our TIPPERS testbed and CFAST simulator to further evaluate the FireDeX approach.

Thank you